

# Control and Synthesis of Non-Interferent Timed Systems

Gilles Benattar, Franck Cassez, Didier Lime and Olivier H. Roux,

## Abstract

In this paper, we focus on the synthesis of secure timed systems which are modelled as timed automata. The security property that the system must satisfy is a *non-interference* property. Intuitively, non-interference ensures the absence of any causal dependency from a high-level domain to a lower-level domain. Various notions of non-interference have been defined in the literature, and in this paper we focus on *Strong Non-deterministic Non-Interference* (SNNI) and two (bi)simulation based variants thereof (CSNNI and BSNNI). We consider

*timed* non-interference properties for timed systems specified by *timed automata* and we study the two following problems: (1) check whether it is possible to find a sub-system so that it is non-interferent; if yes (2) compute a (largest) sub-system which is non-interferent.

## Index Terms

Non-Interference, Timed Automaton, Safety Timed Games, Control, Synthesis

## I. INTRODUCTION

Modern computing environments allow the use of programs that are sent or fetched from different sites. Such programs may deal with secret information such as private data (of a user) or classified data (of an organization). One of the basic concerns in such a context is to ensure that the programs do not leak sensitive data to a third party, either maliciously or inadvertently. This is often called *secrecy*.

In an environment with two parties, *information flow analysis* defines secrecy as: “high-level information never flows into low-level channels”. Such a definition is referred to as a *non-interference* property, and may capture any causal dependency between high-level and low-level behaviors.

We assume that there are two users and the set of actions of the system  $S$  is partitioned into  $\Sigma_h$  (high-level actions) and  $\Sigma_l$  (low-level actions). The non-interference properties we focus on are strong non-deterministic non-interference (SNNI), cosimulation-based strong non-deterministic non-interference (CSNNI) and bisimulation-based strong non-deterministic non-interference (BSNNI). The *non-interference verification problem*, for a given system  $S$ , consists in checking whether  $S$  is non-interferent. It is worth noticing that non-interferent properties are out of the scope of the common safety/liveness classification of system properties [1].

There is a large body of works on the use of static analysis techniques to guarantee information flow policies. A general overview can be found in [2]. Verification of information flow security properties [1], [3] can be applied to the analysis of cryptographic protocols where many uniform and concise characterizations of information flow security properties (*e.g.* confidentiality, authentication, non-repudiation or anonymity) in terms of non-interference have been proposed. For example, the Needham-Schroeder protocol can be proved insecure by defining the security property using SNNI [4], and other examples of the use of non-interference in computer systems and protocols for checking security properties can be found in [5], [6], [7], [8].

In case a system is not non-interferent, it is interesting to investigate how and if it can be rendered non-interferent.

This is the scope of this paper where we consider the problem of *synthesizing* non-interferent timed systems. In contrast to verification, the *non-interference synthesis problem* assumes the system is *open*, *i.e.*, we can restrict the behaviors of  $S$ : some events, from a particular set  $\Sigma_c \subseteq \Sigma_l \cup \Sigma_h$ , of  $S$  can be disabled. The *non-interference control problem* for a system  $S$  asks the following: “Is there a controller  $C$  s.t.  $C(S)$  is non-interferent?” where  $C(S)$  is “ $S$  controlled by  $C$ ”. The associated *synthesis problem* asks to compute a witness controller  $C$  when one exists.

As mentioned earlier, SNNI is expressive enough for example to prove that the Needham-Schroeder protocol is flawed [4]. Controller synthesis enables one to find automatically the patch(es) to apply to make such a protocol secure. The use of dense-time to model the system clearly gives a more accurate and realistic model for the system and a potential attacker that can measure time.

**Related Work.** In [9] the authors consider the complexity of many non-interference *verification* problems but synthesis is not addressed. In [10] an exponential time decision procedure for checking whether a finite state system satisfies a given Basic

G. Benattar is with ClearSy (Safety Critical Systems Engineering Company) Paris, France.

D. Lime and O. H. Roux are with IRCCyN laboratory, LUNAM Université, Ecole Centrale Nantes, France.

F. Cassez is with National ICT Australia, Sydney, Australia.

Security Predicate (BSP) is presented but the synthesis problem is not addressed. Recently supervisory control for opacity property has been studied in [11], [12], [13] in the untimed setting. Opacity is undecidable for timed systems [14] and thus the associated control problem is undecidable as well. In [15] the controller synthesis problem for non-interference properties is addressed for untimed systems. In [16], supervisory control to enforce Intransitive non-interference for three level security systems is proposed in the untimed setting.

The non-interference synthesis problem for dense-time systems specified by timed automata was first considered in [17]. The non-interference property considered in [17] is the *state* non-interference property, which is less demanding than the one we consider here. This paper extends the results of [18] about *SNNI control problems* for timed systems: Section V addresses the SNNI control problem for timed systems and is a detailed presentation of the result of [18] with proofs of the theorems that were unpublished. Sections III and IV are new and the latter provides a new result, Theorem 2. Section VI addresses the CSNNI and BSNNI control problems for timed systems and also contains new results: Theorems 9, 10, 11 and Propositions 4 and 5.

**Our Contribution.** In this paper, we first exhibit a class *dTA* of timed automata for which the SNNI verification problem is decidable. The other main results are: (1) we prove that deciding whether there is a controller  $C$  for a timed automaton  $A$  such that (s.t. in the following)  $C(A)$  is SNNI, is decidable for the previous class *dTA*; (2) we reduce the SNNI controller synthesis problem to solving a sequence of *safety timed games*; (3) we show that there is not always a most permissive controller for CSNNI and BSNNI; (4) we prove that the control problem for CSNNI is decidable for the class *dTA* and that the CSNNI controller synthesis problem for *dTA* reduces to the SNNI controller synthesis problem. We also give the theoretical complexities of these problems.

**Organization of the paper.** Section II recalls the basics of timed automata, timed languages and some results on safety timed games. Section III gives the definition of the non-interference properties we are interested in. Section IV addresses the verification of non-interference properties in the timed setting. Section V gives the definition of the non-interference synthesis problem and presents the main result: we show that there is a largest subsystem which is SNNI and this subsystem is effectively computable. Section VI addresses the control problem and controller synthesis problem for CSNNI and BSNNI properties. Finally, we conclude in Section VII.

## II. PRELIMINARIES

Let  $\mathbb{R}_+$  be the set of non-negative reals and  $\mathbb{N}$  the set of integers. Let  $X$  be a finite set of positive real-valued variables called *clocks*. A valuation of the variables in  $X$  is a function  $X \rightarrow \mathbb{R}_+$ , that can be written as a vector of  $\mathbb{R}_+^X$ . We let  $\vec{0}_X$  be the valuation s.t.  $\vec{0}_X(x) = 0$  for each  $x \in X$  and use  $\vec{0}$  when  $X$  is clear from the context. Given a valuation  $v$  and  $R \subseteq X$ ,  $v[R \mapsto 0]$  is the valuation s.t.  $v[R \mapsto 0](x) = v(x)$  if  $x \notin R$  and 0 otherwise. An atomic constraint (over  $X$ ) is of the form  $x \bowtie c$ , with  $x \in X$ ,  $\bowtie \in \{<, \leq, =, \geq, >\}$  and  $c \in \mathbb{N}$ . A (convex) formula is a conjunction of atomic constraints.  $\mathcal{C}(X)$  is the set of convex formulas. Given a valuation  $v$  (over  $X$ ) and a formula  $\gamma$  over  $X$ ,  $\gamma(v)$  is the truth value, in  $\mathbb{B} = \{\text{true}, \text{false}\}$ , of  $\gamma$  when each symbol  $x$  in  $\gamma$  is replaced by  $v(x)$ . If  $t \in \mathbb{R}_+$ , we let  $v + t$  be the valuation s.t.  $(v + t)(x) = v(x) + t$ . We let  $|V|$  be the cardinality of the set  $V$ .

Let  $\Sigma$  be a finite set,  $\varepsilon \notin \Sigma$  and  $\Sigma^\varepsilon = \Sigma \cup \{\varepsilon\}$ . A *timed word*  $w$  over  $\Sigma$  is a sequence  $w = (\delta_0, a_0)(\delta_1, a_1) \cdots (\delta_n, a_n)$  s.t.  $(\delta_i, a_i) \in \mathbb{R}_+ \times \Sigma$  for  $0 \leq i \leq n$  where  $\delta_i$  represents the amount of time elapsed<sup>1</sup> between  $a_{i-1}$  and  $a_i$ .  $T\Sigma^*$  is the set of timed words over  $\Sigma$ . We denote by  $uv$  the *concatenation* of two timed words  $u$  and  $v$ . As usual  $\varepsilon$  is also the empty word s.t.  $(\delta_1, \varepsilon)(\delta_2, a) = (\delta_1 + \delta_2, a)$ : this means that language-wise, we can always eliminate the  $\varepsilon$  action by taking into account its time interval in the next visible action. Given a timed word  $w \in T\Sigma^*$  and  $L \subseteq \Sigma$  the *projection* of  $w$  over  $L$  is denoted by  $\text{proj}_L(w)$  and is defined by  $\text{proj}_L(w) = (\delta_0, b_0)(\delta_1, b_1) \cdots (\delta_n, b_n)$  with  $b_i = a_i$  if  $a_i \in L$  and  $b_i = \varepsilon$  otherwise. The *untimed projection* of  $w$ ,  $\text{Untimed}(w)$ , is the word  $a_0 a_1 \cdots a_n$  of  $\Sigma^*$ .

A *timed language* is a subset of  $T\Sigma^*$ . Let  $L$  be a timed language, the untimed language of  $L$  is  $\text{Untimed}(L) = \{v \in \Sigma^* \mid \exists w \in L \text{ s.t. } v = \text{Untimed}(w)\}$ .

**Definition 1** (Timed Transition System (TTS)). A timed transition system (TTS) is a tuple  $\mathcal{S} = (Q, q_0, \Sigma^\varepsilon, \rightarrow)$  where  $Q$  is a set of states,  $q_0$  is the initial state,  $\Sigma$  a finite alphabet of actions,  $\rightarrow \subseteq Q \times \Sigma^\varepsilon \cup \mathbb{R}_+ \times Q$  is the transition relation. We use the notation  $q \xrightarrow{e} q'$  if  $(q, e, q') \in \rightarrow$ . Moreover, TTS should satisfy the classical time-related conditions where  $d, d' \in \mathbb{R}_{\geq 0}$ : i) *time determinism*:  $(q \xrightarrow{d} q') \wedge (q \xrightarrow{d} q'') \Rightarrow (q' = q'')$ , ii) *time additivity*:  $(q \xrightarrow{d} q') \wedge (q' \xrightarrow{d'} q'') \Rightarrow (q \xrightarrow{d+d'} q'')$ , iii) *null delay*:  $\forall q : q \xrightarrow{0} q$ , and iv) *time continuity*:  $(q \xrightarrow{d} q') \Rightarrow (\forall d' \leq d, \exists q'', q \xrightarrow{d'} q'')$ .

A run  $\rho$  of  $\mathcal{S}$  from  $q_0$  is a finite sequence of transitions  $\rho = q_0 \xrightarrow{e_1} q_1 \xrightarrow{e_2} \cdots \xrightarrow{e_n} q_n$  s.t.  $(q_i, e_i, q_{i+1}) \in \rightarrow$  for  $0 \leq i \leq n-1$ . We denote by  $\text{last}(\rho)$  the last state of the sequence i.e., the state  $q_n$ . We let  $\text{Runs}(q, \mathcal{S})$  be the set of runs from  $q$  in  $\mathcal{S}$  and  $\text{Runs}(\mathcal{S}) = \text{Runs}(q_0, \mathcal{S})$ . We write  $q \xRightarrow{\varepsilon} q'$  if there is a run  $q \xrightarrow{\varepsilon} \cdots \xrightarrow{\varepsilon} q'$  from  $q$  to  $q'$  i.e.,  $\xRightarrow{\varepsilon} \stackrel{\text{def}}{=} (\xrightarrow{\varepsilon})^*$ . Given  $a \in \Sigma \cup \mathbb{R}_+$ , we define  $\xRightarrow{a} \stackrel{\text{def}}{=} \xRightarrow{\varepsilon} \xrightarrow{a} \xRightarrow{\varepsilon}$ . We write  $q_0 \xrightarrow{*} q_n$  if there is a run from  $q_0$  to  $q_n$ . The set of *reachable* states

<sup>1</sup>For  $i = 0$  this is the amount of time since the system started.

in  $\text{Runs}(\mathcal{S})$  is  $\text{Reach}(\mathcal{S}) = \{q \mid q_0 \xrightarrow{*} q\}$ . Each run can be written in a normal form where delay and discrete transitions alternate i.e.,  $\rho = q_0 \xrightarrow{\delta_0, e_0} q_1 \xrightarrow{\delta_1, e_1} \dots \xrightarrow{\delta_n, e_n} q_{n+1} \xrightarrow{\delta} q'_{n+1}$ . The trace of  $\rho$  is  $\text{trace}(\rho) = (\delta_0, e_0)(\delta_1, e_1) \dots (\delta_n, e_n)$ .

**Definition 2** (Timed automata (TA)). A timed automaton (TA) is a tuple  $A = (Q, q_0, X, \Sigma^\varepsilon, E, \text{Inv})$  where:  $q_0 \in Q$  is the initial location;  $X$  is a finite set of positive real-valued clocks;  $\Sigma^\varepsilon$  is a finite set of actions;  $E \subseteq Q \times \mathcal{C}(X) \times \Sigma^\varepsilon \times 2^X \times Q$  is a finite set of edges. An edge  $(q, \gamma, a, R, q')$  goes from  $q$  to  $q'$ , with the guard  $\gamma \in \mathcal{C}(X)$ , the action  $a$  and the reset set  $R \subseteq X$ ;  $\text{Inv} : Q \rightarrow \mathcal{C}(X)$  is a function that assigns an invariant to any location; we require that the atomic formulas of an invariant are of the form  $x \bowtie c$  with  $\bowtie \in \{<, \leq\}$ .

A finite (or untimed) automaton  $A = (Q, q_0, \Sigma^\varepsilon, E)$  is a special kind of timed automaton with  $X = \emptyset$ , and consequently all the guards and invariants are vacuously true. A timed automaton  $A$  is *deterministic* if for  $(q_1, \gamma, a, R, q_2), (q_1, \gamma', a, R', q'_2) \in E, \gamma \wedge \gamma' \neq \text{false} \Rightarrow q_2 = q'_2$  and  $R = R'$ . We recall that timed automata cannot always be determinized (i.e., find a deterministic TA which accepts the same language as a non-deterministic one, see [19]), and moreover, checking whether a timed automaton is determinizable is undecidable [20].

**Definition 3** (Semantics of Timed automata). The semantics of a timed automaton  $A = (Q, q_0, X, \Sigma^\varepsilon, E, \text{Inv})$  is the TTS  $\mathcal{S}^A = (S, s_0, \Sigma^\varepsilon, \rightarrow)$  with  $S = Q \times (\mathbb{R}^+)^X$ ,  $s_0 = (q_0, \vec{0})$ , and  $\rightarrow$  defined as follows:

$$(q, v) \xrightarrow{a} (q', v') \text{ iff } \exists (q, \gamma, a, R, q') \in E \text{ such that } \begin{cases} \gamma(v) = \text{true} \\ v' = v[R \mapsto 0] \\ \text{Inv}(q')(v') = \text{true} \end{cases}$$

$$(q, v) \xrightarrow{\delta} (q, v') \text{ iff } \begin{cases} v' = v + \delta \\ \forall \delta', 0 \leq \delta' \leq \delta, \\ \text{Inv}(q)(v + \delta') = \text{true} \end{cases}$$

If  $s = (q, v)$  is a state of  $\mathcal{S}^A$ , we denote by  $s + \delta$  the (only) state reached after  $\delta$  time units, i.e.,  $s + \delta = (q, v + \delta)$ . The sets of runs of  $A$  is defined as  $\text{Runs}(A) = \text{Runs}(\mathcal{S}^A)$  where  $\mathcal{S}^A$  is the semantics of  $A$ . A timed word  $w \in T\Sigma^*$  is *generated* by  $A$  if  $w = \text{trace}(\rho)$  for some  $\rho \in \text{Runs}(A)$ . The timed language generated by  $A$ ,  $\mathcal{L}(A)$ , is the set of timed words generated by  $A$ .

**Definition 4** (Language equivalence). Two automata  $A$  and  $B$  are language equivalent, denoted by  $A \approx_{\mathcal{L}} B$ , if  $\mathcal{L}(A) = \mathcal{L}(B)$  i.e., they generate the same set of timed words.

**Definition 5** (Simulation). Let  $\mathcal{T}_1 = (S_1, s_0^1, \Sigma^\varepsilon, \rightarrow_1)$ ,  $\mathcal{T}_2 = (S_2, s_0^2, \Sigma^\varepsilon, \rightarrow_2)$  be two TTS. Let  $\mathcal{R} \subseteq S_1 \times S_2$  be a relation s.t.  $\mathcal{R}$  is total for  $S_2$ .  $\mathcal{R}$  is a weak simulation of  $\mathcal{T}_2$  by  $\mathcal{T}_1$  iff:

- 1)  $s_0^1 \mathcal{R} s_0^2$ ,
- 2)  $\forall (s, p) \in S_1 \times S_2$ , such that  $s \mathcal{R} p$ :
  - If  $p \xrightarrow{\varepsilon}_2 p'$  then  $\exists s'$  such that  $s \xrightarrow{\varepsilon}_1 s'$  and  $s' \mathcal{R} p'$ ,
  - $\forall a \in \Sigma \cup \mathbb{R}_+$ , if  $p \xrightarrow{a}_2 p'$  then  $\exists s'$  such that  $s \xrightarrow{a}_1 s'$  and  $s' \mathcal{R} p'$ .

$\mathcal{T}_1$  weakly simulates  $\mathcal{T}_2$  if there exists a weak simulation  $\mathcal{R}$  of  $\mathcal{T}_2$  by  $\mathcal{T}_1$  and we note  $\mathcal{T}_1 \sqsubseteq_{\mathcal{W}} \mathcal{T}_2$ . Let  $A_1$  and  $A_2$  be two timed automata, we say that  $A_1$  weakly simulates  $A_2$  if the semantics of  $A_1$  weakly simulates the semantics of  $A_2$ , and we note  $A_1 \sqsubseteq_{\mathcal{W}} A_2$ .

**Definition 6** (Cosimulation). Two timed automata  $A_1$  and  $A_2$  are co-similar iff  $A_1 \sqsubseteq_{\mathcal{W}} A_2$  and  $A_2 \sqsubseteq_{\mathcal{W}} A_1$ . We note  $A_1 \approx_{\mathcal{CW}} A_2$ .

**Definition 7** (Bisimulation). Two timed automata  $A_1$  and  $A_2$  are bisimilar iff there exists a simulation  $\mathcal{R}$  of  $A_2$  by  $A_1$  such that  $\mathcal{R}^{-1}$  is a weak simulation of  $A_1$  by  $A_2$ . We note  $A_1 \approx_{\mathcal{W}} A_2$ .

Note that when no  $\varepsilon$  transition exists, we obtain *strong* versions of similarity and bisimilarity.

**Definition 8** (Product of timed automata). Let  $A_1 = (Q_1, q_{01}, X_1, \Sigma^\varepsilon, E_1, \text{Inv}_1)$  and  $A_2 = (Q_2, q_{02}, X_2, \Sigma^\varepsilon, E_2, \text{Inv}_2)$  be two TA with  $X_1 \cap X_2 = \emptyset$ . Let  $\Sigma_a \subseteq \Sigma$ . The synchronized product of  $A_1$  and  $A_2$  w.r.t.  $\Sigma_a$ , is the timed automaton  $A_1 \times_{\Sigma_a} A_2 = (Q_1 \times Q_2, (q_{01}, q_{02}), X_1 \cup X_2, \Sigma^\varepsilon, E, \text{Inv})$  where  $E$  is defined as follows:

- $((q_1, q_2), \gamma_1 \wedge \gamma_2, a, R_1 \cup R_2, (q'_1, q'_2)) \in E$  if  $a \in \Sigma_a$ ,  $(q_1, \gamma_1, a, R_1, q'_1) \in E_1$  and  $(q_2, \gamma_2, a, R_2, q'_2) \in E_2$ ;
- $((q_1, q_2), \gamma, a, R, (q'_1, q'_2)) \in E$  if  $a \in \Sigma \setminus \Sigma_a$  and  $\begin{cases} (q_1, \gamma, a, R, q'_1) \in E_1 \text{ and } q'_2 = q_2 \\ \text{or } (q_2, \gamma, a, R, q'_2) \in E_2 \text{ and } q'_1 = q_1 \end{cases}$

and where  $\text{Inv}((q_1, q_2)) = \text{Inv}_1(q_1) \wedge \text{Inv}_2(q_2)$ .

It means that synchronization occurs only for actions in  $\Sigma_a$ . When it is clear from the context we omit the subscript  $\Sigma_a$  in  $\times_{\Sigma_a}$ .

Moreover, in the sequel we will use two operators on TA: the first one gives an *abstracted* automaton and simply hides a set of labels  $L \subseteq \Sigma$ . Given a TA  $A = (Q, q_0, X, \Sigma^\varepsilon, E, Inv)$  and  $L \subseteq \Sigma$  we define the TA  $A/L = (Q, q_0, X, (\Sigma \setminus L)^\varepsilon, E_L, Inv)$  where  $(q, \gamma, a, R, q') \in E_L \iff (q, \gamma, a, R, q') \in E$  for  $a \in \Sigma \setminus L$  and  $(q, \gamma, \varepsilon, R, q') \in E_L \iff (q, \gamma, a, R, q') \in E$  for  $a \in L \cup \{\varepsilon\}$ . The *restricted* automaton cuts transitions labeled by the letters in  $L \subseteq \Sigma$ : Given a TA  $A = (Q, q_0, X, \Sigma, E, Inv)$  and  $L \subseteq \Sigma$  we define the TA  $A \setminus L = (Q, q_0, X, \Sigma \setminus L, E_L, Inv)$  where  $(q, \gamma, a, R, q') \in E_L \iff (q, \gamma, a, R, q') \in E$  for  $a \in \Sigma \setminus L$ .

We will also use some results on safety control for timed games which have been introduced and solved in [21].

**Definition 9** (Timed Game Automaton (TGA)). A Timed Game Automaton (TGA)  $A = (Q, q_0, X, \Sigma, E, Inv)$  is a timed automaton with its set of actions  $\Sigma$  partitioned into controllable ( $\Sigma_c$ ) and uncontrollable ( $\Sigma_u$ ) actions.

Let  $A$  be a TGA and  $Bad \subseteq Q \times \mathbb{R}_+^X$  be the set of bad states to avoid.  $Bad$  can be written  $\cup_{1 \leq i \leq k} (\ell_i, Z_i)$ , with each  $Z_i$  defined as a conjunction of formulas of  $\mathcal{C}(X)$  and each  $\ell_i \in Q$ . The *safety control problem* for  $(A, Bad)$  is: decide whether there is a controller to constantly avoid  $Bad$ . Let  $\lambda$  be a fresh special symbol not in  $\Sigma^\varepsilon$  denoting the action “do nothing”.

A *controller*  $C$  for  $A$  is a partial function from  $Runs(A)$  to  $2^{\Sigma_c \cup \{\lambda\}}$ . We require that  $\forall \rho \in Runs(A)$ , if  $a \in C(\rho) \cap \Sigma_c$  then  $last(\rho) \xrightarrow{a} (q', v')$  for some  $(q', v')$  and if  $\lambda \in C(\rho)$  then  $last(\rho) \xrightarrow{\delta} (q', v')$  for some  $\delta > 0$ . A controller  $C$  is *state-based* or *memoryless* whenever  $\forall \rho, \rho' \in Runs(A)$ ,  $last(\rho) = last(\rho')$  implies that  $C(\rho) = C(\rho')$ .

**Remark 1.** We assume a controller gives a set of actions that are enabled which differs from standard definitions [21] where a controller only gives one action. Nevertheless for safety timed games, one computes a most permissive controller (if there is one) which gives for each state the largest set of actions which are safe. It follows that any reasonable (e.g., Non-Zeno) sub-controller of this most permissive controller avoids the set of bad states.

$C(A)$  defines “ $A$  supervised/restricted by  $C$ ” and is inductively defined by its set of runs:

- $(q_0, \vec{0}) \in Runs(C(A))$ ,
- if  $\rho \in Runs(C(A))$  and  $\rho \xrightarrow{e} s' \in Runs(A)$ , then  $\rho \xrightarrow{e} s' \in Runs(C(A))$  if one of the following three conditions holds:
  - 1)  $e \in \Sigma_u$ ,
  - 2)  $e \in \Sigma_c \cap C(\rho)$ ,
  - 3)  $e \in \mathbb{R}_+$  and  $\forall \delta$  s.t.  $0 \leq \delta < e$ ,  $last(\rho) \xrightarrow{\delta} last(\rho) + \delta \wedge \lambda \in C(\rho \xrightarrow{\delta} last(\rho) + \delta)$ .

$C(A)$  can also be viewed as a TTS where each state is a run of  $A$  and the transitions are given by the previous definition.  $C$  is a *winning controller* for  $(A, Bad)$  if  $Reach(C(A)) \cap Bad = \emptyset$ . For safety timed games, the results are the following [21], [22]:

- it is (EXPTIME-complete to decide whether there is a winning controller for a safety game  $(A, Bad)$ ;
- in case there is one, there is a *most permissive* controller which is memoryless on the region graph of the TGA  $A$ . This most permissive controller can be represented by a TA. This also means that the set of runs of  $C(A)$  is itself the semantics of a timed automaton, that can be effectively built from  $A$ .

### III. FORMAL DEFINITIONS OF NON-INTERFERENCE PROPERTIES

In the sequel, we will consider Timed Automata defined on an set of actions  $\Sigma = \Sigma_l \cup \Sigma_h$  with  $\Sigma_l \cap \Sigma_h = \emptyset$ , where  $\Sigma_h$  are the *high level* actions and  $\Sigma_l$  the *low level* actions. In order to define the different classes of non interference properties on an automaton  $A$ , we are going to compare  $A \setminus \Sigma_h$  and  $A/\Sigma_h$  w.r.t. different criteria.

#### A. Strong Non-Deterministic Non-Interference (SNNI)

The property *Strong Non-Deterministic Non-Interference* (SNNI) has been introduced by Focardi and Gorrieri in [1] as a *trace-based* generalization of non-interference for concurrent systems. SNNI has been extended to timed models in [17].

**Definition 10.** A timed automaton  $A$  is SNNI iff  $A \setminus \Sigma_h \approx_{\mathcal{L}} A/\Sigma_h$

Since finite automata are timed automata with no clocks, the definition also applies to finite automata.

Moreover, as  $\mathcal{L}(A \setminus \Sigma_h) \subseteq \mathcal{L}(A/\Sigma_h)$ , we can give a simple characterization of the SNNI property:

**Proposition 1.** A timed automaton  $A$  is SNNI iff  $\mathcal{L}(A/\Sigma_h) \subseteq \mathcal{L}(A \setminus \Sigma_h)$ .

**Example 1.** Let us consider the automaton  $A_a$  of figure 1(a) with  $\Sigma_h = \{h\}$  and  $\Sigma_l = \{\ell\}$ . This automaton is not SNNI, because  $\mathcal{L}(A \setminus \Sigma_h) = \varepsilon$  whereas  $\mathcal{L}(A/\Sigma_h) = \ell$ . The automaton  $A_b$  is SNNI.

As demonstrated by the following examples 2 and 3, a timed automaton  $A$  can be non SNNI whereas its untimed underlying automaton is SNNI and  $A$  can be SNNI whereas its untimed underlying automaton is not.

**Example 2.** Let us consider the timed automaton  $A_g$  of figure 2(a), with  $\Sigma_h = \{h\}$  and  $\Sigma_l = \{\ell\}$ . It is not SNNI since  $(2.5, \ell)$  is accepted by  $A_g/\Sigma_h$  but not by  $A_g \setminus \Sigma_h$ . Its untimed underlying automaton  $A_h$  is SNNI since  $\mathcal{L}(A_h \setminus \Sigma_h) = \{\ell\} = \mathcal{L}(A_h/\Sigma_h)$ .

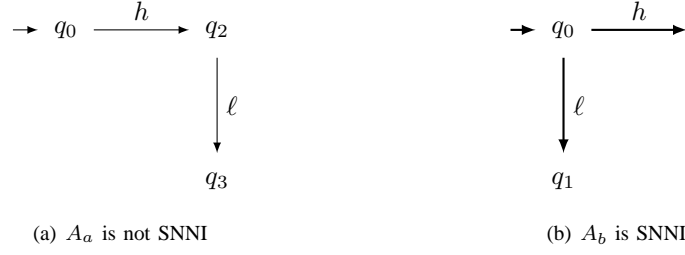


Fig. 1. Examples for the SNNI property

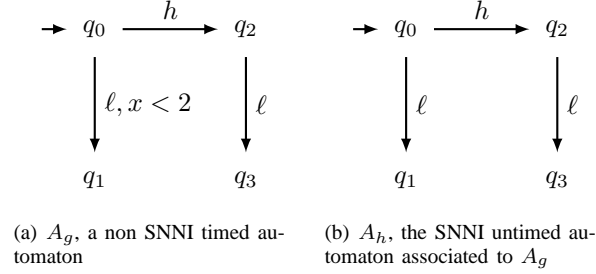


Fig. 2. A non SNNI timed automaton and its untimed underlying automaton which is SNNI

**Example 3.** Let us consider the timed automaton  $A_j$  of figure 3(a), with  $\Sigma_h = \{h\}$  et  $\Sigma_l = \{\ell_1, \ell_2\}$ . It is SNNI, since  $\mathcal{L}(A_j \setminus \Sigma_h) = \mathcal{L}(A_j / \Sigma_h)$ . Its untimed underlying automaton  $A_k$  is not SNNI since  $\ell_1 \cdot \ell_2$  is accepted by  $A_k / \Sigma_h$  but not by  $A_k \setminus \Sigma_h$ .

**Example 4 (SNNI).** Figure 4 gives examples of systems  $A(k)$  which are SNNI and not SNNI depending on the value of integer  $k$ . The high-level actions are  $\Sigma_h = \{h\}$  and the low-level actions are  $\Sigma_l = \{l\}$ .  $(\delta, l)$  with  $1 \leq \delta < 2$  is a trace of  $A(1) / \Sigma_h$  but not of  $A(1) \setminus \Sigma_h$  and so,  $A(1)$  is not SNNI.  $A(2)$  is SNNI as we can see that  $A(2) / \Sigma_h \approx_{\mathcal{L}} A(2) \setminus \Sigma_h$ .

Finally since SNNI is based on language equivalence, we have the following lemma:

**Lemma 1.** If  $A' \approx_{\mathcal{L}} A$ , then  $A$  is SNNI  $\Leftrightarrow A'$  is SNNI.

*Proof:* First  $\mathcal{L}(A / \Sigma_h) = \text{proj}_{\Sigma_l}(\mathcal{L}(A)) = \text{proj}_{\Sigma_l}(\mathcal{L}(A')) = \mathcal{L}(A' / \Sigma_h)$ . Second,  $\mathcal{L}(A \setminus \Sigma_h) = \mathcal{L}(A) \cap T\Sigma_l^* = \mathcal{L}(A') \cap T\Sigma_l^* = \mathcal{L}(A' \setminus \Sigma_h)$ . ■

### B. Cosimulation Strong Non-Deterministic Non-Interference (CSNNI)

The *Cosimulation Strong Non-Deterministic Non-Interference* (CSNNI) property has been introduced in [17], and is based on *cosimulation*.

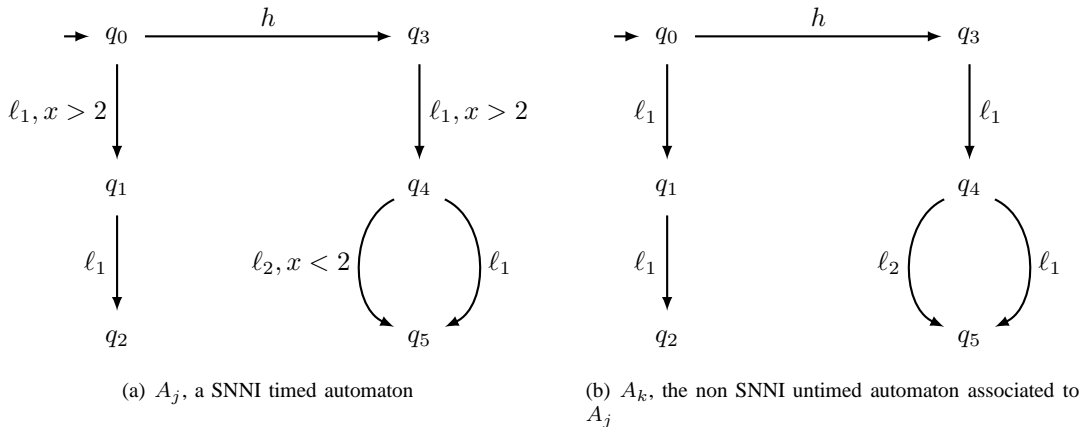


Fig. 3. A SNNI timed automaton and its untimed underlying automaton which is non SNNI.

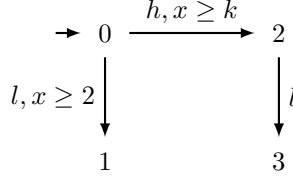
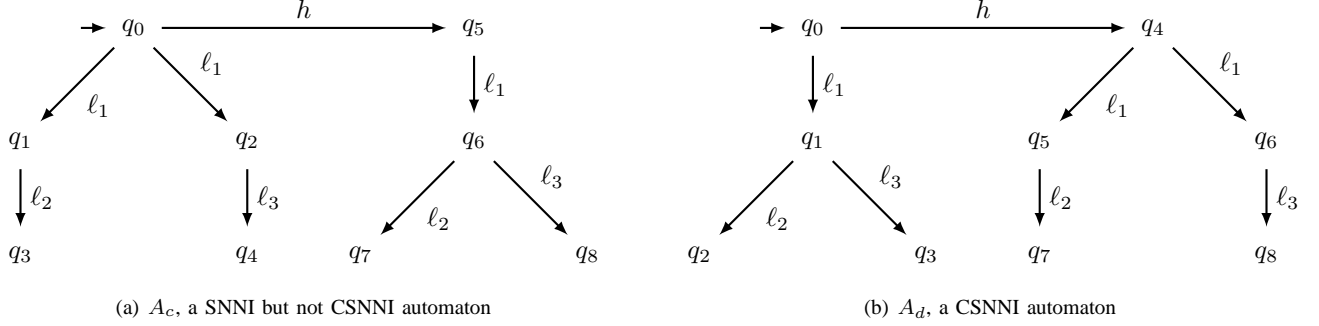
Fig. 4. Automaton  $A(k)$ 

Fig. 5. CSNNI is stronger than SNNI

**Definition 11.** A timed automaton  $A$  is CSNNI iff  $A \setminus \Sigma_h \approx_{\mathcal{CW}} A / \Sigma_h$ .

Since  $A / \Sigma_h \sqsubseteq_{\mathcal{W}} A \setminus \Sigma_h$ , we can give a simple characterization of CSNNI:

**Proposition 2.** A timed Automaton  $A$  is CSNNI iff  $A \setminus \Sigma_h \sqsubseteq_{\mathcal{W}} A / \Sigma_h$ .

By restricting the class of timed automata considered, we obtain the following result.

**Example 5.** Let us consider the automaton  $A_c$  of figure 5(a) with  $\Sigma_h = \{h\}$  and  $\Sigma_l = \{\ell_1, \ell_2, \ell_3\}$ .  $A_c$  is SNNI but is not CSNNI, because no state of  $A_c \setminus \Sigma_l$  can simulate the state  $q_6$ . The automaton  $A_d$  of figure 5(b) is CSNNI. The state  $q_1$  of  $A_d \setminus \Sigma_l$  simulates the states  $q_5$  and  $q_6$ .

We complete this subsection by comparing SNNI and CSNNI. Given two timed automata  $A_1, A_2$ ,  $A_1 \sqsubseteq_{\mathcal{W}} A_2$  implies  $\mathcal{L}(A_2) \subseteq \mathcal{L}(A_1)$ . CSNNI is thus stronger than SNNI as for each timed automaton  $A$ ,  $A \setminus \Sigma_h \sqsubseteq_{\mathcal{W}} A / \Sigma_h$  implies  $\mathcal{L}(A / \Sigma_h) \subseteq \mathcal{L}(A \setminus \Sigma_h)$ .

The converse holds when  $A \setminus \Sigma_h$  is deterministic:

**Lemma 2.** If  $A \setminus \Sigma_h$  is deterministic, then  $A$  is SNNI implies  $A$  is CSNNI.

*Proof:* As emphasized before, given two timed automata  $A_1, A_2$ ,  $A_1 \sqsubseteq_{\mathcal{W}} A_2$  implies  $\mathcal{L}(A_2) \subseteq \mathcal{L}(A_1)$ . If  $A_1$  is deterministic, then  $\mathcal{L}(A_2) \subseteq \mathcal{L}(A_1)$  implies  $A_1 \sqsubseteq_{\mathcal{W}} A_2$ . To obtain the result it suffices to take  $A_1 = A \setminus \Sigma_h$  and  $A_2 = A / \Sigma_h$ . ■

### C. Bisimulation Strong Non-Deterministic Non-Interference (BSNNI)

The Bisimulation Strong Non-Deterministic Non-Interference (BSNNI) property has been introduced in [1] and is based on bisimulation.

**Definition 12.** A timed automaton  $A$  is BSNNI iff  $A \setminus \Sigma_h \approx_{\mathcal{W}} A / \Sigma_h$

The automaton  $A_f$  of figure 6(b) is BSNNI. Bisimulation is stronger than cosimulation and we have for all timed automaton  $A$ , if  $A$  is BSNNI then  $A$  is CSNNI (and thus  $A$  is SNNI).

As the following example demonstrates, there exists an automaton which is CSNNI and not BSNNI.

**Example 6.** Let us consider the automaton  $A_e$  of figure 6(a) with  $\Sigma_h = \{h\}$  et  $\Sigma_l = \{\ell\}$ . This automaton is deterministic and SNNI, and therefore by lemma 2, it is CSNNI. However, it is not BSNNI, since the state  $q_2$  of  $A_e \setminus \Sigma_h$  has no bisimilar state in  $A_e \setminus \Sigma_h$ .

## IV. VERIFICATION OF NON-INTERFERENCE PROPERTIES FOR TIMED AUTOMATA

In this section we settle the complexity of non-interference verification problems for timed automata.

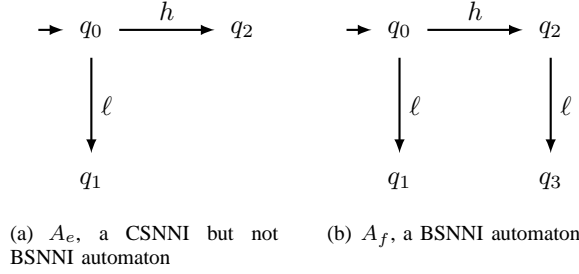


Fig. 6. BSNNI is stronger than CSNNI

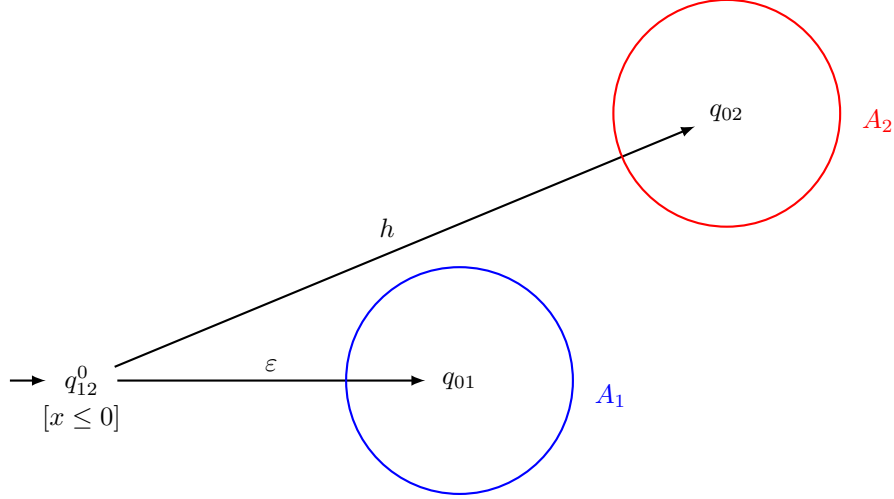


Fig. 7. The timed automaton  $A_{12}$

#### A. SNNI verification

The SNNI verification problem (SNNI-VP), asks to check whether a system  $A$  is SNNI.

For timed automata, this problem has been proved to be *undecidable* in [17] and the proof is based on the fact that language containment for TA is undecidable [19]. However, if we consider the subclass of timed automata  $A$  such that  $A \setminus \Sigma_h$  is *deterministic*, then the problem becomes decidable. In the sequel, we called *dTA* the class of timed automata  $A$  such that  $A \setminus \Sigma_h$  is deterministic.

**Theorem 1.** *The SNNI-VP is PSPACE-complete for dTA.*

*Proof:* Let  $A_1$  and  $A_2$  be two timed automata. Checking whether  $\mathcal{L}(A_2) \subseteq \mathcal{L}(A_1)$  with  $A_1$  a deterministic TA is PSPACE-complete [19]. Checking  $\mathcal{L}(A/\Sigma_h) \subseteq \mathcal{L}(A \setminus \Sigma_h)$  can thus be done in PSPACE if  $A \setminus \Sigma_h$  is deterministic. Using Proposition 1, it follows that SNNI-VP is PSPACE-easy for *dTA*.

For PSPACE-hardness, we reduce the language inclusion problem  $\mathcal{L}(A_2) \subseteq \mathcal{L}(A_1)$ , with  $A_1$  a deterministic TA, to the SNNI-VP. Let  $A_1 = (Q_1, q_{01}, X_1, \Sigma, E_1, Inv_1)$  be a deterministic TA and  $A_2 = (Q_2, q_{02}, X_2, \Sigma, E_2, Inv_2)$  a TA<sup>2</sup>. We let  $h \notin \Sigma$  be a fresh letter,  $x \notin X_1 \cup X_2$  be a fresh clock and define  $A_{12} = (\{q_{12}^0\} \cup Q_1 \cup Q_2, q_{01}, X_1 \cup X_2 \cup \{x\}, \Sigma^\varepsilon \cup \{h\}, E_{12}, Inv_{12})$  be the timed automaton defined (as shown in figure 7) as follows:

- the transition relation  $E_{12}$  contains  $E_1 \cup E_2$  and the additional transitions  $(q_{12}^0, true, h, \emptyset, q_{02})$  and  $(q_{12}^0, true, \varepsilon, \emptyset, q_{01})$ ;
- $Inv_{12}(q) = Inv_i(q)$  if  $q \in Q_i, i \in \{1, 2\}$ , and  $Inv_{12}(q_{12}^0) = [x \leq 0]$ .

We let  $\Sigma_l = \Sigma$  and  $\Sigma_h = \{h\}$ . We prove that  $A_{12}$  is SNNI iff  $\mathcal{L}(A_2) \subseteq \mathcal{L}(A_1)$ . This is easily established as:

$$\begin{aligned}
 A_{12} \text{ is SNNI} & \text{ iff } \mathcal{L}(A_{12}/\Sigma_h) \subseteq \mathcal{L}(A_{12} \setminus \Sigma_h) & [\text{Proposition 1}] \\
 & \text{ iff } \mathcal{L}(A_1) \cup \mathcal{L}(A_2) \subseteq \mathcal{L}(A_1) \\
 & \text{ iff } \mathcal{L}(A_2) \subseteq \mathcal{L}(A_1).
 \end{aligned}$$

Thus the SNNI-VP is PSPACE-complete for *dTA*. ■

<sup>2</sup>We assume that  $Q_1 \cap Q_2 = \emptyset$  and  $X_1 \cap X_2 = \emptyset$ .

For non-deterministic finite automata  $A_1$  and  $A_2$ , checking language inclusion  $\mathcal{L}(A_1) \subseteq \mathcal{L}(A_2)$  is PSPACE-complete [23]. Then, using the same proof with  $A_1$  being a non deterministic finite automaton, It follows that:

**Corollary 1.** *The SNNI-VP is PSPACE-complete for non-deterministic finite automata.*

Moreover, when  $A_2$  is a deterministic finite automaton, language containment can be checked in PTIME and thus we have the following corollary:

**Corollary 2.** *For finite automata belonging to dTA, the SNNI-VP is PTIME.*

The table I summarizes the results on the complexity of the SNNI-VP.

	Timed Automata	Finite Automata
$A \setminus \Sigma_h$ is deterministic (dTA)	PSPACE-complete (Theorem 1)	PTIME (Corollary 2)
General Case	Undecidable [17]	PSPACE-complete (Corollary 1)

TABLE I  
COMPLEXITY IF SNNI-VP

### B. Verification of CSNNI and BSNNI properties

BSNNI-VP and CSNNI-VP are decidable for timed automata [17] since simulation and bisimulation are decidable. For finite automata, the complexity of BSNNI-VP and CSNNI-VP is known to be PTIME [15]. We settle here the complexity of those problems for timed automata.

**Theorem 2.** *The CSNNI-VP and BSNNI-VP are EXPTIME-complete for Timed Automata.*

*Proof:* Strong timed bisimilarity and simulation pre-order are both EXPTIME-complete for timed automata. The EXPTIME-hardness is established in [24] where it is shown that any relation between simulation pre-order and bisimilarity is EXPTIME-hard for Timed Automata.

The EXPTIME-easiness for strong timed bisimulation was established in [25] and for simulation pre-order in [26].

To establish EXPTIME-completeness for CSNNI-VP and BSNNI-VP, we show that these problems are equivalent to their counterparts for timed automata.

To do this, we use the automata  $A_1, A_2$  and  $A_{12}$  already defined in the proof of Theorem 1.

We show that:  $A_1$  simulates  $A_2$  iff  $A_{12}$  is CSNNI.

Assume  $A_1$  simulates  $A_2$ . There exists a relation  $\mathcal{R}$  s.t. : 1)  $(q_{01}, \vec{0}_{X_1})\mathcal{R}(q_{01}, \vec{0}_{X_1})$  and 2) for each state  $(s_2, \vec{x}_2)$ , there exists  $(s_1, \vec{x}_1)$  s.t.  $(s_2, \vec{x}_2)\mathcal{R}(s_1, \vec{x}_1)$ , and whenever  $(s_2, \vec{x}_2) \xrightarrow{a} (s'_2, \vec{x}'_2)$  for  $a \in \Sigma \cup \mathbb{R}_+$ , then  $(s_1, \vec{x}_1) \xrightarrow{a} (s'_1, \vec{x}'_1)$  and  $(s'_2, \vec{x}'_2)\mathcal{R}(s'_1, \vec{x}'_1)$ .

We define a relation  $\mathcal{R}'$  for each  $(\ell, \vec{x}_1 \vec{x}_2 x)$  of  $A_{12}/\Sigma_h$  to a state  $(\ell', \vec{x}'_1 \vec{x}'_2 x')$  of  $A_{12} \setminus \Sigma_h$  as follows:

- if  $\ell = q_{12}^0$  then  $(\ell, \vec{x}_1 \vec{x}_2 x)\mathcal{R}'(\ell, \vec{x}'_1 \vec{x}'_2 x')$ ;
- if  $\ell \in Q_1$ , then  $(\ell, \vec{x}_1 \vec{x}_2 x)\mathcal{R}'(\ell, \vec{x}'_1 \vec{x}'_2 x')$ ;
- if  $\ell \in Q_2$ , then  $(\ell, \vec{x}_1 \vec{x}_2 x)\mathcal{R}'(\ell', \vec{x}'_1 \vec{x}'_2 x')$  iff  $(\ell, \vec{x}_2)\mathcal{R}(\ell', \vec{x}'_1)$ ;

$\mathcal{R}'$  is a simulation of  $A_{12}/\Sigma_h$  by  $A_{12} \setminus \Sigma_h$ :

- the initial states of the two TA are in relation;
- assume  $(s, \vec{x}_1 \vec{x}_2 x) \xrightarrow{a} A_{12}/\Sigma_h (s', \vec{x}'_1 \vec{x}'_2 x')$ ; If  $s \in \{q_{12}^0\} \cup Q_1$  then clearly it is simulated by the same state in  $A_{12} \setminus \Sigma_h$ . Otherwise, if  $s \in Q_2$ , then there exists a state  $(\ell', \vec{x}'_1 \vec{x}'_2 x')$  in  $A_{12} \setminus \Sigma_h$  s.t.  $(s, \vec{x}_1 \vec{x}_2 x)\mathcal{R}'(\ell', \vec{x}'_1 \vec{x}'_2 x')$ : by definition of  $\mathcal{R}'$  we can take any  $(s', \vec{x}'_1 \vec{x}'_2 x')$  with  $(s, \vec{x}_2)\mathcal{R}(s', \vec{x}'_1)$ . It is easy to see that because  $A_1$  can simulate  $A_2$  from there on,  $\mathcal{R}'$  is indeed a simulation relation. Thus  $A_{12}/\Sigma_h$  and  $A_{12} \setminus \Sigma_h$  are co-similar by Proposition 2.

Now assume conversely that there is a simulation  $\mathcal{R}'$  of  $A_{12}/\Sigma_h$  by  $A_{12} \setminus \Sigma_h$ . We can define a simulation relation of  $A_2$  by  $A_1$  as follows: each state  $(s, \vec{x}_1 \vec{x}_2 x)$  with  $s \in Q_2$  of  $A_{12}/\Sigma_h$  is simulated by a state  $(s', \vec{x}'_1 \vec{x}'_2 x')$  with  $s' \in Q_1$ . We then define  $\mathcal{R}$  by  $(s, \vec{x}_2)\mathcal{R}(s', \vec{x}'_1)$ . Again it is easy to see that  $\mathcal{R}$  is a simulation relation.

It follows that CSNNI is EXPTIME-complete.

Now assume that  $A_1$  and  $A_2$  are bisimilar. We can define the relation  $\mathcal{R}'$  exactly as above and this time it is a weak bisimulation between  $A_{12} \setminus \Sigma_h$  and  $A_{12}/\Sigma_h$ .

If  $A_{12}$  is BSNNI, the bisimulation relation  $\mathcal{R}'$  between  $A_{12} \setminus \Sigma_h$  and  $A_{12}/\Sigma_h$  induces a bisimulation relation  $\mathcal{R}$  between  $A_1$  and  $A_2$ : it suffices to build  $\mathcal{R}$  as the restriction of  $\mathcal{R}'$  between states with a discrete component in  $Q_1$  and a discrete component in  $Q_2$ .

As checking bisimulation between TA is also EXPTIME-complete, the EXPTIME-completeness of BSNNI-VP for TA follows. ■

The table II summarize the results on the verification of the CSNNI and BSNNI properties.



	Timed Automata	Finite Automata
CSNNI-VP	EXPTIME-C (Theorem 2)	PTIME [15]
BSNNI-VP	EXPTIME-C (Theorem 2)	PTIME [15]

TABLE II  
RESULTS FOR CSNNI-VP AND BSNNI-VP

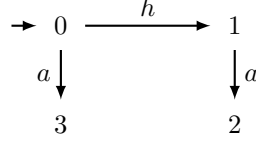


Fig. 8. Automaton  $D$

## V. THE SNNI CONTROL PROBLEM

The previous non-interference verification problem, consists in *checking* whether an automaton  $A$  has the non-interference property. If the answer is “no”, one has to investigate why the non-interference property is not true, modify  $A$  and check the property again. In contrast to the verification problem, the synthesis problem indicates whether there is a way of restricting the behavior of users to ensure a given property. Thus we consider that only some actions in the set  $\Sigma_c$ , with  $\Sigma_c \subseteq \Sigma_h \cup \Sigma_l$ , are controllable and can be disabled. We let  $\Sigma_u = \Sigma \setminus \Sigma_c$  denote the actions that are uncontrollable and thus cannot be disabled. Note that, contrary to [15], we release the constraint  $\Sigma_c = \Sigma_h$ . The motivations for this work are many fold. Releasing  $\Sigma_c = \Sigma_h$  is interesting in practice because it enables one to specify that an action from  $\Sigma_h$  cannot be disabled (a service must be given), while some actions of  $\Sigma_l$  can be disabled. We can view actions of  $\Sigma_l$  as capabilities of the low-level user (e.g., pressing a button), and it thus makes sense to prevent the user from using the button for instance by disabling/hiding it temporarily.

Recall that a *controller*  $C$  for  $A$  gives for each run  $\rho$  of  $A$  the set  $C(\rho) \in 2^{\Sigma_c \cup \{\lambda\}}$  of actions that are enabled after this particular run. The *SNNI-Control Problem* (SNNI-CP) we are interested in is the following:

*Is there a controller  $C$  s.t.  $C(A)$  is SNNI ?* (SNNI-CP)

The *SNNI-Controller Synthesis Problem* (SNNI-CSP) asks to compute a witness when the answer to the SNNI-CP is “yes”.

### A. Preliminary Remarks

First we motivate our definition of controllers which are mappings from  $Runs(A)$  to  $2^{\Sigma_c \cup \{\lambda\}}$ . The common definition of a controller in the literature is a mapping from  $Runs(A)$  to  $\Sigma_c \cup \{\lambda\}$ . Indeed, for the safety (or reachability) control problem, one can compute a mapping  $M : Runs(A) \rightarrow 2^{\Sigma_c \cup \{\lambda\}}$  (most permissive controller), and a controller  $C$  ensures the safety goal iff  $C(\rho) \in M(\rho)$ . This implies that any sub-controller of  $M$  is a good controller. This is not the case for SNNI, even for finite automata, as the following example shows.

**Example 7.** Let us consider the automaton  $D$  of Figure 8 with  $\Sigma_c = \{a, h\}$ . The largest sub-system of  $D$  which is SNNI is  $D$  itself. Disabling  $a$  from state 0 will result in an automaton which is not SNNI.

We are thus interested in computing the largest (if there is such) sub-system of  $A$  that we can control which is SNNI. Second, in our definition we allow a controller to forbid any controllable action. In contrast, in the literature, a controller should ensure some liveness and never block the system. In the context of security property, it makes sense to disable everything if the security policy cannot be enforced otherwise. This makes the SNNI-CP easy for finite automata.

### B. SNNI-VP versus SNNI-CP

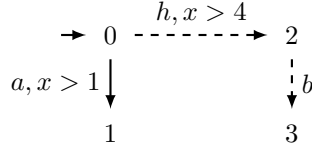
SNNI-CP is harder than SNNI-VP since SNNI-VP reduces to SNNI-CP by taking  $\Sigma_c = \emptyset$ . Note that this is not true if we restrict to the subclass of control where  $\Sigma_c = \Sigma_h$ . Indeed, in this case SNNI-CP is always true (and then decidable) since the controller which forbid all controllable transitions make the system SNNI.

We then have the following theorem:

**Theorem 3.** For general Timed Automata, SNNI-CP and SNNI-CSP are undecidable.

*Proof:* SNNI-CP obviously reduces to SNNI-CSP. SNNI-VP reduces to SNNI-CP by taking  $\Sigma_c = \emptyset$ . SNNI-VP is undecidable for non-deterministic Timed Automata. ■

We will now show that SNNI-CP reduces to the SNNI-VP for finite automata.

Fig. 9. The Automaton  $H$ 

**Theorem 4.** *For finite automata, the SNNI-CP is PSPACE-Complete.*

*Proof:* The proof consists in proving that if a finite automaton can be restricted to be SNNI, then disabling all the  $\Sigma_c$  actions is a solution. Thus the SNNI-CP reduces to the SNNI-VP and the result follows.

As time is not taken into account in untimed automaton, we can have  $C(\rho) = \emptyset$  for finite automaton (for general timed automaton, this would mean that we block the time.) The proof of the theorem consists in proving that if a finite automaton can be restricted to be SNNI, then disabling all the  $\Sigma_c$  actions is a solution. Let  $C_V$  be the controller defined by  $C_V(\rho) = \emptyset$ . We prove the following: if  $C$  is a controller s.t.  $C(A)$  is SNNI, then  $C_V(A)$  is SNNI.

Assume a finite automaton  $D$  is SNNI. Let  $e \in \Sigma_h \cup \Sigma_l$  and let  $\mathcal{L}_e$  be the set of words containing at least one  $e$ . Depending on the type of  $e$  we have:

- if  $e \in \Sigma_l$ , then  $\mathcal{L}((D \setminus \{e\}) \setminus \Sigma_h) = \mathcal{L}(D \setminus \Sigma_h) \setminus \mathcal{L}_e$  and as  $D$  is SNNI, it is also equal to  $\mathcal{L}(D \setminus \Sigma_h) \setminus \mathcal{L}_e = \mathcal{L}((D \setminus \{e\}) \setminus \Sigma_h)$ ;
- if  $e \in \Sigma_h$ ,  $\mathcal{L}((D \setminus \{e\}) \setminus \Sigma_h) \subseteq \mathcal{L}(D \setminus \Sigma_h) = \mathcal{L}(D \setminus \Sigma_h) = \mathcal{L}((D \setminus \{e\}) \setminus \Sigma_h)$ .

So, if  $D$  is SNNI,  $D \setminus L$  is SNNI,  $\forall L \subseteq \Sigma$ . Since  $\mathcal{L}(C_V(D)) = \mathcal{L}(D \setminus \Sigma_c)$ , if  $D$  is SNNI, then  $D \setminus \Sigma_c$  is also SNNI and therefore  $C_V(D)$  is SNNI.

Let  $A$  be the TA we want to restrict. Assume there is a controller  $C$  s.t.  $C(A)$  is SNNI.  $C_V(C(A))$  is SNNI so  $C_V(C(A)) = C_V(A)$  is also SNNI which means that  $A \setminus \Sigma_c$  is SNNI. This proves that:  $\exists C$  s.t.  $C(A)$  is SNNI  $\Leftrightarrow A \setminus \Sigma_c$  is SNNI.

It is then equivalent to check that  $A \setminus \Sigma_c$  is SNNI to solve the SNNI-CP for  $A$  and this can be done in PSPACE. PSPACE-hardness comes from the reduction of SNNI-VP to SNNI-CP, by taking  $\Sigma_c = \emptyset$ . ■

Moreover since the SNNI-CP reduces to the SNNI-VP for finite automata, and from corollary 2 we have the following result:

**Corollary 3.** *For finite automata belonging to dTA, the SNNI-CP is PTIME.*

We will now show that Theorem 4 does not hold for timed automata as the following example demonstrates.

**Example 8.** Figure 9 gives an example of a timed automaton  $H$  with high-level actions  $\Sigma_h = \{h\}$  and low-level actions  $\Sigma_l = \{a, b\}$ .

Assume  $\Sigma_c = \{a\}$ . Notice that  $H \setminus \Sigma_c$  is not SNNI. Let the state based controller  $C$  be defined by:  $C(0, x) = \{a, \lambda\}$  when  $H$  is in state  $(0, x)$  with  $x < 4$ ; and  $C(0, x) = \{a\}$  when  $x = 4$ . Then  $C(H)$  is SNNI. In this example, when  $x = 4$  we prevent time from elapsing by forcing the firing of  $a$  which indirectly disables action  $h$ . To do this we just have to add an invariant  $[x \leq 4]$  to location 0 of  $H$  and this cuts out the dashed transitions rendering  $C(H)$  SNNI.

### C. Algorithms for SNNI-CP and SNNI-CSP

In this section we first prove that the SNNI-CP is EXPTIME-hard for dTA. Then we give an EXPTIME algorithm to solve the SNNI-CP and SNNI-CSP.

**Theorem 5.** *For dTA, the SNNI-CP is EXPTIME-Hard.*

*Proof:* The safety control problem for TA is EXPTIME-hard [27]. In the proof of this theorem, T.A. Henzinger and P.W. Kopke use timed automata where the controller chooses an action and the environment resolves non-determinism. The hardness proof reduces the halting problem for alternating Turing Machines using polynomial space to a safety control problem. In our framework, we use TA with controllable and uncontrollable actions. It is not difficult to adapt the hardness proof of [27] to TA which are deterministic w.r.t.  $\Sigma_c$  actions and non deterministic w.r.t.  $\Sigma_u$  actions. As  $\Sigma_u$  transitions can never be disabled (they act only as spoiling actions), we can use a different label for each uncontrollable transition without altering the result in our definition of the safety control problem. Hence: the safety control problem as defined in section II is EXPTIME-hard for deterministic TA (with controllable and uncontrollable transitions). This problem can be reduced to the safety control problem of TA with only one state *bad*. We can now reduce the safety control problem for deterministic TA which is EXPTIME-hard to the SNNI control problem on dTA. Let  $A = (Q \cup \{bad\}, q_0, X, \Sigma_c \cup \Sigma_u, E, Inv)$  be a TGA, with  $\Sigma_c$  (resp.  $\Sigma_u$ ) the set of controllable (resp. uncontrollable) actions, and *bad* a location to avoid. We define  $A'$  by adding to  $A$  two uncontrollable transitions:  $(bad, \text{true}, h, \emptyset, q_h)$  and  $(q_h, \text{true}, l, \emptyset, q_l)$  where  $q_h$  and  $q_l$  are fresh locations with invariant *true*.  $l$  and  $h$  are two fresh uncontrollable actions in  $A'$ . We now define  $\Sigma_h = \{h\}$  and  $\Sigma_l = \Sigma_c \cup \Sigma_u \cup \{l\}$  for  $A'$ . By definition of  $A'$ , for

any controller  $C$ , if location  $Bad$  is not reachable in  $C(A')$ , then the actions  $h$  and then  $l$  can not be fired. Thus if there is controller for  $C$  for  $A$  which avoids  $Bad$ , the same controller  $C$  renders  $A'$  SNNI. Now if there is a controller  $C'$  s.t.  $C'(A')$  is SNNI, it must never enable  $h$ : otherwise a (untimed) word  $w.h.l$  would be in  $Untimed(\mathcal{L}(C'(A')/\Sigma_h))$  but as no untimed word containing an  $l$  can be in  $Untimed(\mathcal{L}(C'(A')/\Sigma_h))$ , and thus  $C'(A')$  would not be SNNI. Notice that it does not matter whether we require the controllers to be non blocking (mappings from  $Runs(A)$  to  $2^{\Sigma_c \cup \{\lambda\}} \setminus \emptyset$ ) or not as the reduction holds in any case. ■

To compute the most permissive controller (and we will also prove there is one), we build a safety game and solve a safety control problem. It may be necessary to iterate this procedure. Of course, we restrict our attention to TA in the class  $dTA$  for which the SNNI-VP is decidable.

Let  $A = (Q, q_0, X, \Sigma_h \cup \Sigma_l, E, Inv)$  be a TA s.t.  $A \setminus \Sigma_h$  is deterministic. The idea of the reduction follows from the following remark: we want to find a controller  $C$  s.t.  $\mathcal{L}(C(A) \setminus \Sigma_h) = \mathcal{L}(C(A)/\Sigma_h)$ . For any controller  $C$  we have  $\mathcal{L}(C(A) \setminus \Sigma_h) \subseteq \mathcal{L}(C(A)/\Sigma_h)$  because each run of  $C(A) \setminus \Sigma_h$  is a run of  $C(A)/\Sigma_h$ . To ensure SNNI we must have  $\mathcal{L}(C(A)/\Sigma_h) \subseteq \mathcal{L}(A \setminus \Sigma_h)$ : indeed,  $A \setminus \Sigma_h$  is the largest language that can be generated with no  $\Sigma_h$  actions, so a necessary condition for enforcing SNNI is  $\mathcal{L}(C(A)/\Sigma_h) \subseteq \mathcal{L}(A \setminus \Sigma_h)$ . The controller  $C(A)$  indicates what must be pruned out in  $A$  to ensure the previous inclusion. Our algorithm thus proceeds as follows: we first try to find a controller  $C^1$  which ensures that  $\mathcal{L}(C^1(A)/\Sigma_h) \subseteq \mathcal{L}(A \setminus \Sigma_h)$ . If  $\mathcal{L}(C^1(A)/\Sigma_h) = \mathcal{L}(A \setminus \Sigma_h)$  then  $C^1$  is the most permissive controller that enforces SNNI. It could be that what we had to prune out to ensure  $\mathcal{L}(C^1(A)/\Sigma_h) \subseteq \mathcal{L}(A \setminus \Sigma_h)$  does not render  $C^1(A)$  SNNI. In this case we may have to iterate the previous procedure on the new system  $C^1(A)$ .

We first show how to compute  $C^1$ . As  $A \setminus \Sigma_h$  is deterministic, we can construct  $A_2 = (Q \cup \{q_{bad}\}, q_0^2, X_2, \Sigma_h \cup \Sigma_l, E_2, Inv_2)$  which is a copy of  $A$  (with clock renaming) with  $q_{bad}$  being a fresh location and s.t.  $A_2$  is a *complete* (i.e.,  $\mathcal{L}(A_2) = T\Sigma^*$ ) version of  $A \setminus \Sigma_h$  ( $A_2$  is also deterministic). We write  $last_2(w)$  the state  $(q, v)$  reached in  $A_2$  after reading a timed word  $w \in T\Sigma^*$ .  $A_2$  has the property that  $w \in \mathcal{L}(A \setminus \Sigma_h)$  if the state reached in  $A_2$  after reading  $w$  is not in  $Bad$  with  $Bad = \{(q_{bad}, v) \mid v \in \mathbb{R}_+^X\}$ .

**Fact 1.** Let  $w \in T\Sigma^*$ . Then  $w \notin \mathcal{L}(A \setminus \Sigma_h) \iff last_2(w) \in Bad$ .

We now define the product  $A_p = A \times_{\Sigma_l} A_2$  and the set of bad states,  $Bad^\otimes$  of  $A_p$  to be the set of states where  $A_2$  is in  $Bad$ .  $\rightarrow_p$  denotes the transition relation of the semantics of  $A_p$  and  $s_p^0$  the initial state of  $A_p$ . When it is clear from the context we omit the subscript  $p$  in  $\rightarrow_p$ .

**Lemma 3.** Let  $w \in \mathcal{L}(A)$ . Then there is a run  $\rho \in Runs(A_p)$  s.t.  $\rho = s_p^0 \xrightarrow{w} s$  with  $s \in Bad^\otimes$  iff  $\text{proj}_{\Sigma_l}(w) \notin \mathcal{L}(A \setminus \Sigma_h)$ .

The proof follows easily from Fact 1. Given a run  $\rho$  in  $Runs(A_p)$ , we let  $\rho_{|1}$  be the projection of the run  $\rho$  on  $A$  (uniquely determined) and  $\rho_{|2}$  be the unique run<sup>3</sup> in  $A_2$  whose trace is  $\text{proj}_{\Sigma_l}(\text{trace}(\rho))$ . The following Theorem proves that any controller  $C$  s.t.  $C(A)$  is SNNI can be used to ensure that  $Bad^\otimes$  is not reachable in the game  $A_p$ :

**Lemma 4.** Let  $C$  be a controller for  $A$  s.t.  $C(A)$  is SNNI. Let  $C^\otimes$  be a controller on  $A_p$  defined by  $C^\otimes(\rho') = C(\rho'_{|1})$ . Then,  $\text{Reach}(C^\otimes(A_p)) \cap Bad^\otimes = \emptyset$ .

*Proof:* First  $C^\otimes$  is well-defined because  $\rho'_{|1}$  is uniquely defined. Let  $C$  be a controller for  $A$  s.t.  $C(A)$  is SNNI. Assume  $\text{Reach}(C^\otimes(A_p)) \cap Bad^\otimes \neq \emptyset$ . By definition, there is a run  $\rho'$  in  $Runs(C^\otimes(A_p))$  such that:

$$\begin{aligned} \rho' &= ((q_0, q_0^2), (\vec{0}, \vec{0})) \xrightarrow{e_1} ((q_1, q_1'), (v_1, v_1')) \xrightarrow{e_2} \dots \xrightarrow{e_n} ((q_n, q_n'), (v_n, v_n')) \\ &\xrightarrow{e_{n+1}} ((q_{n+1}, q_{n+1}'), (v_{n+1}, v_{n+1}')) \end{aligned}$$

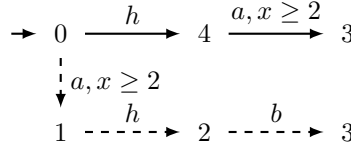
with  $((q_{n+1}, q_{n+1}'), (v_{n+1}, v_{n+1}')) \in Bad^\otimes$  and we can assume  $(q_i', v_i') \notin Bad$  for  $1 \leq i \leq n$  (and  $q_0^2 \notin Bad$ ). Let  $\rho = \rho'_{|1}$  and  $w = \text{proj}_{\Sigma_l}(\text{trace}(\rho')) = \text{proj}_{\Sigma_l}(\text{trace}(\rho))$ . We can prove (1):  $\rho \in Runs(C(A))$  and (2):  $w \notin \mathcal{L}(C(A) \setminus \Sigma_h)$ . (1) directly follows from the definition of  $C^\otimes$ . This implies that  $w \in \mathcal{L}(C(A)/\Sigma_h)$ . (2) follows from Lemma 3. By (1) and (2) we obtain that  $w \in \mathcal{L}(C(A)/\Sigma_h) \setminus \mathcal{L}(C(A) \setminus \Sigma_h)$  i.e.,  $\mathcal{L}(C(A)/\Sigma_h) \not\subseteq \mathcal{L}(C(A) \setminus \Sigma_h)$  and so  $C(A)$  does not have the SNNI property which is a contradiction. Hence  $\text{Reach}(C^\otimes(A_p)) \cap Bad^\otimes = \emptyset$ . ■

If we have a controller which solves the safety game  $(A_p, Bad^\otimes)$ , we can build a controller which ensures that  $\mathcal{L}(C(A)/\Sigma_h) \subseteq \mathcal{L}(A \setminus \Sigma_h)$ . Notice that as emphasized before, this does not necessarily ensure that  $C(A)$  is SNNI.

**Lemma 5.** Let  $C^\otimes$  be a controller for  $A_p$  s.t.  $\text{Reach}(C^\otimes(A_p)) \cap Bad^\otimes = \emptyset$ . Let  $C(\rho) = C^\otimes(\rho')$  if  $\rho'_{|1} = \rho$ .  $C$  is well-defined and  $\mathcal{L}(C(A)/\Sigma_h) \subseteq \mathcal{L}(A \setminus \Sigma_h)$ .

*Proof:* Let  $\rho = (q_0, \vec{0}) \xrightarrow{e_1} (q_1, v_1) \xrightarrow{e_2} \dots \xrightarrow{e_n} (q_n, v_n)$  be a run of  $A$ . Since  $A_2$  is deterministic and complete there is exactly one run  $\rho' = ((q_0, q_0), (\vec{0}, \vec{0})) \xrightarrow{e_1} ((q_1, q_1'), (v_1, v_1')) \xrightarrow{e_2} \dots \xrightarrow{e_n} ((q_n, q_n'), (v_n, v_n'))$  in  $A_p$  s.t.  $\rho'_{|1} = \rho$ . So  $C$  is well-defined. Now, assume there is some  $w \in \mathcal{L}(C(A)/\Sigma_h) \setminus \mathcal{L}(A \setminus \Sigma_h)$ . Then, there is a run  $\rho$  in  $Runs(C(A)) \subseteq Runs(A)$  s.t.  $\text{proj}_{\Sigma_l}(\text{trace}(\rho)) = w$ , there is a unique run  $\rho$  in  $Runs(A_p)$  s.t.  $\rho_{|1} = \rho$  and  $\text{trace}(\rho') = w$ . First by Lemma 3,  $last(\rho') \in$

<sup>3</sup>Recall that  $A_2$  is deterministic.

Fig. 10. The Automaton  $K$ 

$Bad^\otimes$ . Second, this run  $\rho'$  is in  $Runs(C^\otimes(A_p))$  because of the definition of  $C$ . Hence  $Reach(C^\otimes(A_p)) \cap Bad^\otimes \neq \emptyset$  which is a contradiction. ■

It follows that if  $C^\otimes$  is the most permissive controller for  $A_p$  then  $C(A)$  is a timed automaton (and can be effectively computed) because the most permissive controller for safety timed games is memoryless. More precisely, let  $RG(A_p)$  be the region graph of  $A_p$ .  $C$  is memoryless on  $RG(A_p \setminus \Sigma_h)$  because  $A_2$  is deterministic. The memory required by  $C$  is at most  $RG(A \setminus \Sigma_h)$  on the rest of the region graph of  $RG(A_p)$ .

Assume the safety game  $(A_p, Bad^\otimes)$  can be won and  $C^\otimes$  is the most permissive controller. Let  $C$  be the controller obtained using Lemma 5. Controller  $C$  ensures that  $\mathcal{L}(C(A)/\Sigma_h) \subseteq \mathcal{L}(A/\Sigma_h)$ . But as the following example shows, it may be the case that  $C(A)$  is not SNNI.

**Example 9.** Consider the TA  $K$  of Figure 10 with  $\Sigma_h = \{h\}$  and  $\Sigma_c = \{a\}$ .

We can compute  $C(K)$  from  $C^\otimes$  which satisfies  $Reach(C^\otimes(K \times_{\Sigma_1} K_2)) \cap Bad^\otimes = \emptyset$ , and is given by the sub-automaton of  $K$  with the plain arrows.  $C(K)$  is obviously not SNNI. For the example of  $A(1)$  in Figure 4, if we compute  $C$  in the same manner, we obtain  $C(A(1)) = A(2)$  and moreover  $\mathcal{L}(C(A(1))/\Sigma_h) = \mathcal{L}(A(1)/\Sigma_h)$ . And then the most permissive sub-system which is SNNI is given by  $C(A(1)) = A(2)$  (the guard  $x \geq 1$  of  $A(1)$  is strengthened).

The example of Figure 10 shows that computing the most permissive controller on  $A_p$  is not always sufficient. Actually, we may have to iterate the computation of the most permissive controller on the reduced system  $C(A)$ .

**Lemma 6.** Consider the controller  $C$  as defined in Lemma 5. If  $C(A) \setminus \Sigma_h \approx_{\mathcal{L}} A \setminus \Sigma_h$  then  $C(A)$  is SNNI.

*Proof:* If  $C(A) \setminus \Sigma_h \approx_{\mathcal{L}} A \setminus \Sigma_h$ , then,  $\mathcal{L}(C(A)/\Sigma_h) \subseteq \mathcal{L}(A/\Sigma_h) = \mathcal{L}(C(A) \setminus \Sigma_h)$ . As  $\mathcal{L}(C(A) \setminus \Sigma_h) \subseteq \mathcal{L}(C(A)/\Sigma_h)$  is always true,  $\mathcal{L}(C(A)/\Sigma_h) = \mathcal{L}(C(A) \setminus \Sigma_h)$  and so,  $C(A)$  is SNNI. ■

Let  $\perp$  be the symbol that denotes non controllability (or the non existence of a controller). We inductively define the sequence of controllers  $C^i$  and timed automata  $A^i$  as follows:

- let  $C^0$  be the controller defined by  $C^0(\rho) = 2^{\Sigma_c \cup \{\lambda\}}$  and  $A^0 = C^0(A) = A$ ;
- Let  $A_p^i = A^i \times_{\Sigma_1} A_2^i$  and  $C_{i+1}^\otimes$  be the most permissive controller for the safety game  $(A_p^i, Bad_i^\otimes)$  ( $\perp$  if no such controller exists). We use the notation  $Bad_i^\otimes$  because this set depends on  $A_2^i$ . We define  $C^{i+1}$  using Lemma 5:  $C^{i+1}(\rho) = C_{i+1}^\otimes(\rho')$  if  $\rho'_1 = \rho$ . Let  $A^{i+1} = C^{i+1}(A^i)$ .

By Lemma 6, if  $C^{i+1}(A^i) \setminus \Sigma_h \approx_{\mathcal{L}} A^i \setminus \Sigma_h$  then  $C^{i+1}(A^i)$  is SNNI. Therefore this condition is a sufficient condition for the termination of the algorithm defined above:

**Lemma 7.** There exists an index  $i \geq 1$  s.t.  $C^i(A^{i-1})$  is SNNI or  $C^i = \perp$ .

*Proof:* We prove that the region graph of  $C^{i+1}(A^i)$  is a sub-graph of the region graph of  $C^1(A^0)$  for  $i \geq 1$ . By Lemma 5 (and the remark following it),  $C^1(A^0)$  is a sub-graph of  $RG(A \times A_2)$ . Moreover  $C^1$  is memoryless on  $A \setminus \Sigma_h$  and requires a memory of less than  $|RG(A \setminus \Sigma_h)|$  on the remaining part. Assume on this part, a node of  $RG(A \times A_2)$  is of the form  $((q, r), k)$  where  $q$  is a location of  $A$  and  $r$  a region of  $A$  and  $k \in \{1, |RG(A \setminus \Sigma_h)|\}$ .

Assume  $RG(A^k)$  is a sub-graph of  $RG(A^{k-1})$  for  $k \geq 2$  and  $RG(A^{k-1} \setminus \Sigma_h)$  is sub-graph of  $RG(A \setminus \Sigma_h)$ . Using Lemma 5, we can compute  $A^k = C^k(A^{k-1})$  and: (1)  $RG(A^k \setminus \Sigma_h)$  is a sub-graph of  $A^{k-1} \setminus \Sigma_h$  and (2) the memory needed for  $C_k^\otimes$  on the remaining part is less than  $|RG(A^{k-1})|$ . Actually, because  $A^{k-1} \setminus \Sigma_h$  is deterministic, no more memory is required for  $C^k$ . Indeed, the memory corresponds to the nodes of  $A^k \setminus \Sigma_h$ . Thus a node of  $RG(A^k)$  which is not in  $RG(A^k \setminus \Sigma_h)$  is of the form  $((q, r), k, k')$  with  $k = k'$  or  $k' = q_{bad}$ . This implies that  $RG(A^k)$  is a sub-graph of  $RG(A^{k-1})$ .

The most permissive controller  $C_i^\otimes$  will either disable at least one controllable transition of  $A_p^{i-1} \setminus \Sigma_h$  or keep all the controllable transitions of  $A_p^{i-1} \setminus \Sigma_h$ . In the latter case  $A^i \setminus \Sigma_h = A^{i-1} \setminus \Sigma_h$  and otherwise  $|RG(A^i \setminus \Sigma_h)| < |RG(A^{i-1} \setminus \Sigma_h)|$ . This can go on at most  $|RG(A \setminus \Sigma_h)|$  steps. In the end either  $A^i \setminus \Sigma_h = A^{i-1} \setminus \Sigma_h$  and this implies that  $A^i \setminus \Sigma_h \approx_{\mathcal{L}} A^{i-1} \setminus \Sigma_h$  (Lemma 6) or it is impossible to control  $A^{i-1}$  and  $C^i = \perp$ . In any case, our algorithm terminates in less than  $|RG(A)|$  steps. ■

To prove that we obtain the most permissive controller which enforces SNNI, we use the following Lemma:

**Lemma 8.** If  $M$  is a controller such that  $\mathcal{L}(M(A)/\Sigma_h) = \mathcal{L}(M(A) \setminus \Sigma_h)$ , then  $\forall i \geq 0$  and  $\forall \rho \in Runs(A)$ ,  $M(\rho) \subseteq C^i(\rho)$ .

*Proof:* The proof is by induction:

- for  $i = 0$  it holds trivially.
- Assume the Lemma holds for indices up until  $i$ . Thus we have  $Runs(M(A)) \subseteq Runs(A^i)$ . Therefore, we can define  $M$  over  $A^i$  and  $M(A^i)$  is SNNI. By Lemma 4,  $M^\otimes$  is a controller for the safety game  $(A_p^i, Bad_i^\otimes)$ , therefore  $M^\otimes(\rho') \subseteq C_{i+1}^\otimes(\rho')$  because  $C_{i+1}^\otimes$  is the most permissive controller. This implies that  $M(\rho) \subseteq C^{i+1}(\rho)$  by definition of  $C^{i+1}$ . ■

Using Lemma 7, the sequence  $C^i$  converges to a fix-point. Let  $C^*$  denote this fix-point.

**Lemma 9.**  $C^*$  is the most permissive controller for the SNNI-CSP.

*Proof:* Either  $C^* = \perp$  and there is no way of enforcing SNNI (Lemma 4), or  $C^* \neq \perp$  is such that  $\mathcal{L}(C^*(A)/\Sigma_h) = \mathcal{L}(C^*(A) \setminus \Sigma_h)$  by Lemma 5. As for any valid controller  $M$  such that  $\mathcal{L}(M(A)/\Sigma_h) = \mathcal{L}(M(A) \setminus \Sigma_h)$  we have  $M(\rho) \subseteq C^*(\rho)$  for each  $\rho \in Runs(A)$  (Lemma 8) the result follows. ■

Lemma 7 proves the existence of a bound on the number of times we have to solve safety games. For a timed automaton  $A$  in  $dTA$ , let  $|A|$  be the size of  $A$ .

**Lemma 10.** For a  $dTA$   $A$ ,  $C^*$  can be computed in  $O(2^{4 \cdot |A|})$ .

*Proof:* As the proof of Lemma 7 shows, the region graph of  $A^i$  is a sub-graph of the region graph of  $A^1$ ,  $\forall i \geq 1$ , and the algorithm ends in less than  $|RG(A)|$  steps. Computing the most permissive controller for  $A_p^i$  avoiding  $Bad_i^\otimes$  can be done in linear time in the size of the region graph of  $A_p^i$ . As  $RG(A^i)$  is a sub-graph of  $RG(A^1)$ ,  $RG(A_p^i)$  is a sub-graph of  $RG(A_p^1)$ . So we have to solve at most  $|RG(A)|$  safety games of sizes at most  $|RG(A_p^1)|$ . As  $A^1$  is a sub-graph of  $A_p^0 = A^0 \times_{\Sigma_l} A_2^0$ ,  $|RG(A^1)| \leq |RG(A)|^2$ . And as  $A_p^1 = A^1 \times_{\Sigma_l} A_2^1$ ,  $|RG(A_p^1)| \leq |RG(A)|^3$ . So,  $C^*$  can be computed in  $O(|RG(A)| \cdot |RG(A_p^1)|) = O(|RG(A)|^4) = O(2^{4 \cdot |A|})$ . ■

**Theorem 6.** For  $dTA$ , the SNNI-CP and SNNI-CSP are EXPTIME-complete.

For the special case of finite automata we even have:

**Lemma 11.** For finite automata  $C^* = C^2$ .

*Proof:* We know that  $\mathcal{L}(C^2(A) \setminus \Sigma_h) \subseteq \mathcal{L}(C^1(A) \setminus \Sigma_h)$ . Suppose that  $\exists w$  s.t.  $w \in \mathcal{L}(C^1(A) \setminus \Sigma_h)$  and  $w \notin \mathcal{L}(C^2(A) \setminus \Sigma_h)$  ( $w$  cannot not be the empty word). We can assume that  $w = u.l$  with  $u \in \Sigma_l^*$ ,  $l \in \Sigma_l \cap \Sigma_c$  and  $u \in \mathcal{L}(C^1(A) \setminus \Sigma_h)$  and  $u.l \notin \mathcal{L}(C^2(A) \setminus \Sigma_h)$  ( $l$  is the first letter which witnesses the non membership property). If  $l$  had to be pruned in the computation of  $C^2$ , it is because there is a word  $u.l.m$  with  $m \in \Sigma_u^*$  s.t.  $\text{proj}_{\Sigma_l}(u.l.m) \in \mathcal{L}(C^1(A)/\Sigma_h)$  but  $\text{proj}_{\Sigma_l}(u.l.m) \notin \mathcal{L}(C^1(A) \setminus \Sigma_h)$ . But by definition of  $C^1$ ,  $\mathcal{L}(C^1(A)/\Sigma_h) \subseteq \mathcal{L}(A \setminus \Sigma_h)$  (Lemma 5) and thus  $\text{proj}_{\Sigma_l}(u.l.m) \in \mathcal{L}(A \setminus \Sigma_h)$ . As  $u.l \in \Sigma_l^*$ ,  $\text{proj}_{\Sigma_l}(u.l.m) = u.l.\text{proj}_{\Sigma_l}(m)$  and  $\text{proj}_{\Sigma_l}(m) \in \Sigma_u^*$ . Since  $u.l \in \mathcal{L}(C^1(A) \setminus \Sigma_h)$  and  $\text{proj}_{\Sigma_l}(m) \in \Sigma_u^*$ , we have  $u.l.\text{proj}_{\Sigma_l}(m) \in \mathcal{L}(C^1(A) \setminus \Sigma_h)$  which is a contradiction. Thus  $\mathcal{L}(C^2(A) \setminus \Sigma_h) = \mathcal{L}(C^1(A) \setminus \Sigma_h)$  which is our stopping condition by lemma 6 and thus  $C^* = C^2$ . ■

It follows that:

**Theorem 7.** For a finite automaton  $A$  in  $dTA$  (i.e. such that  $A \setminus \Sigma_h$  is deterministic), the SNNI-CSP is PSPACE-complete.

As untimed automata can always be determinized, we can extend our algorithm to untimed automata when  $A \setminus \Sigma_h$  non-deterministic. It suffices to determinize  $A_2^i, i = 1, 2$ :

**Theorem 8.** For a finite automaton  $A$  such that  $A \setminus \Sigma_h$  is non deterministic, the SNNI-CSP can be solved in EXPTIME.

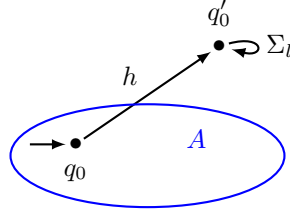
**Proposition 3.** There is a family of finite automata  $(A_i)_{i \geq 0}$  such that: (i) there is a most permissive controller  $D_i^*$  s.t.  $D_i^*(A_i)$  is SNNI and (ii) the memory required by  $D_i^*$  is exponential in the size of  $A_i$ .

*Proof:*

Let  $A$  be a finite automaton over the alphabet  $\Sigma$ . Define the automaton  $A'$  as given by Figure 11. Assume the automaton  $B$  is the sub-automaton of  $A'$  with initial state  $q'_0$ . We take  $\Sigma_h = \{h\} = \Sigma_u$  and  $\Sigma_l = \Sigma = \Sigma_c$ . The most permissive controller  $D$  s.t.  $D(A')$  is SNNI generates the largest sub-language of  $\mathcal{L}(A')$  s.t.  $\mathcal{L}(A' \setminus \Sigma_h) = \mathcal{L}(A'/\Sigma_h)$  and thus it generates  $\mathcal{L}(A) = \mathcal{L}(A' \setminus \Sigma_h)$ .

The controller  $D$  is memoryless on  $A' \setminus \Sigma_h$  as emphasized in Lemma 5. It needs finite memory on the remaining part i.e., on  $B$ . The controller  $D$  on  $B$  gives for each run a set of events of  $\Sigma$  that can be enabled:  $D(q_0 \xrightarrow{h} q'_0 \xrightarrow{w} q'_0) = X$  with  $w \in \Sigma^*$  and  $X \subseteq \Sigma_l$ . As  $B$  is deterministic,  $D$  needs only the knowledge of  $w$  and we can write  $D(hw)$  ignoring the states of  $A'$ . For  $B$  we can even write  $D(w)$  instead of  $D(hw)$ . Define the equivalence relation  $\equiv$  on  $\Sigma^*$  by:  $w \equiv w'$  if  $D(w) = D(w')$ . Denote the class of a word  $w$  by  $[w]$ . Because  $D$  is memory bounded,  $\Sigma_{/\equiv}^*$  is of finite index which is exactly the memory needed by  $D$ .

Thus we can define an automaton  $D_{/\equiv} = (M, m_0, \Sigma, \rightarrow)$  by:  $M = \{[w] \mid w \in \Sigma^*\}$ ,  $m_0 = [\varepsilon]$ , and  $[w] \xrightarrow{a} [wa]$  for  $a \in D(hw)$ .  $D_{/\equiv}$  is an automaton which accepts  $\mathcal{L}(A)$  (and it is isomorphic to  $D(B)$ ) and the size of which is the size of  $D$  because  $B$  has only one state. This automaton is deterministic and thus  $D_{/\equiv}$  is also deterministic and accepts  $\mathcal{L}(A)$ . There is a

Fig. 11. Automaton  $B$ 

family  $(A_i)_{i \geq 0}$  of non-deterministic finite automata, such that the deterministic and language-equivalent automaton of each  $A_i$  requires at least exponential size. For each of these  $A_i$  we construct the controller  $D_{/\equiv}^i$  as described before, and this controller must have at least an exponential size (w.r.t. to  $A_i$ ). This proves the EXPTIME lower bound. ■

In this section we have studied the strong non-deterministic non-interference control problem (SNNI-CP) and control synthesis problem (SNNI-CSP) in the timed setting. The main results we have obtained are: (1) the SNNI-CP can be solved if  $A \setminus \Sigma_h$  can be determinized and is undecidable otherwise; (2) the SNNI-CSP can be solved by solving a finite sequence of safety games if  $A \setminus \Sigma_h$  can be determinized. We have provided an optimal algorithm to solve the SNNI-CP and CSP in this case (although we have not proved a completeness result).

	$A$ Timed Automaton		$A$ Finite Automaton	
	$A \setminus \Sigma_h$ Non-Det.	$A \setminus \Sigma_h$ Det.	$A \setminus \Sigma_h$ Non-Det.	$A \setminus \Sigma_h$ Det.
SNNI-CP	undecidable (Theorem 3)	EXPTIME-C (Theorem 6)	PSPACE-C (Theorem 4)	PTIME (Corollary 3)
SNNI-CSP	undecidable (Theorem 3)	EXPTIME-C (Theorem 6)	EXPTIME (Theorem 8)	PSPACE-C (Theorem 7)

TABLE III  
SUMMARY OF THE RESULTS FOR SNNI-CP AND SNNI-CSP

The summary of the results is given in Table III.

## VI. BSNNI AND CSNNI CONTROL PROBLEMS

In this section, we will show that for more restrictive non-interference properties (CSNNI and BSNNI) the control problem presents a major drawback: in the general case, there is no most permissive controller.

The CSNNI-Control Problem CSNNI-CP (respectively BSNNI-Control Problem BSNNI-CP) we are interested in is the following:

*Is there a controller  $C$  s.t.  $C(A)$  is CSNNI (respectively BSNNI) ?* (CSNNI-CP, BSNNI-CP)

The CSNNI-Controller Synthesis Problem CSNNI-CSP (respectively BSNNI-Controller Synthesis Problem BSNNI-CSP) asks to compute a witness when the answer to the CSNNI-CP (respectively BSNNI-CSP) is “yes”.

### A. CSNNI-CP and CSNNI-CSP

**Theorem 9.** *For finite automata the CSNNI-CP is in PTIME.*

*Proof:*

Let  $A$ , be a finite automaton, we show that there exists a controller  $C$  such that  $C(A)$  is CSNNI if and only if  $A \setminus \Sigma_c$  is CSNNI.

The *if* direction is obvious: the controller  $C_\forall$  that prevents any controllable action from occurring is defined by:  $C_\forall(\rho) = \emptyset$ ,  $\forall \rho \in \text{Runs}(A)$ . It is easy to see that  $C_\forall(A)$  is isomorphic to  $A \setminus \Sigma_c$  and thus bisimilar.

This *only if* direction is proved as follows: let  $A_1$  and  $A_2$  be two finite automata over alphabet  $\Sigma^\varepsilon$  such that  $A_1$  weakly simulates  $A_2$ . Consider  $A'_1 = A_1 \setminus \{e\}$  and  $A'_2 = A_2 \setminus \{e\}$  for  $e \in \Sigma$ . Clearly,  $A'_1$  simulates  $A'_2$  (by definition of the simulation relation).

Therefore, if there exists  $C$  s.t.  $C(A)$  is CSNNI, then so is  $C(A) \setminus \Sigma'$  for any  $\Sigma' \subseteq \Sigma$ . It follows that  $C(A) \setminus \Sigma_c$  must be CSNNI.

The CSNNI-CP reduces to the CSNNI-VP which is PTIME for finite automata. ■

**Theorem 10.** *For the class of deterministic finite automata, the CSNNI-CSP is PSPACE-complete.*

*Proof:* By Lemma 2, for deterministic automata, SNNI is equivalent to CSNNI. Hence the CSNNI-CSP is equivalent to the SNNI-CSP which is PSPACE-complete by Theorem 7. ■

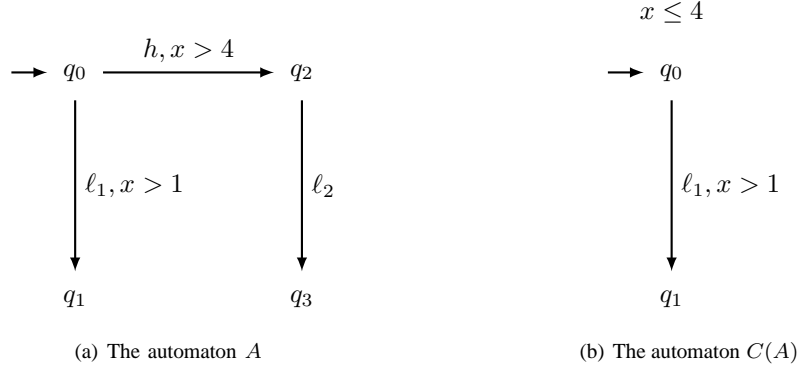
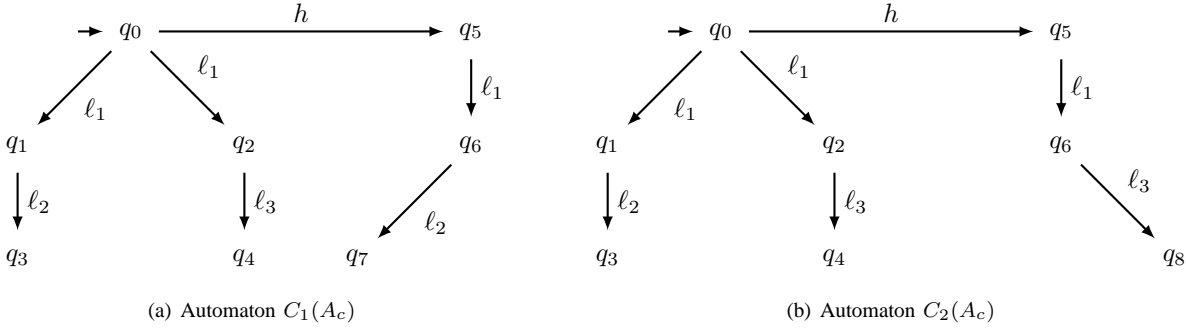


Fig. 12. Counterexample of theorem 9 in timed setting

Fig. 13. Automata  $C_1(A_c)$  and  $C_2(A_c)$ 

In the timed setting, the previous reduction to a verification problem cannot be applied as illustrated by the following example 10.

**Example 10.** Let  $A$  be the deterministic timed automaton given in figure 12(a) with  $\Sigma_l = \{\ell_1, \ell_2\}$ ,  $\Sigma_h = \{h\}$  and  $\Sigma_c = \{\ell_1\}$ .  $A \setminus \Sigma_c$  is neither CSNNI nor SNNI (here SNNI and CSNNI are equivalent since  $A$  is deterministic). However there exists a controller  $C$  such that  $C(A)$  is both CSNNI and SNNI.  $C(A)$  can be given by the timed automaton given in figure 12(b).

However for the timed automata in  $dTA$ , thanks to Lemma 2 and Theorems 6 and 7, we have:

**Theorem 11.** For timed automata in  $dTA$ , the CSNNI-CP and CSNNI-CSP are EXPTIME-complete.

*Proof:* By Lemma 2 the CSNNI-CP/CSNNI-CSP is equivalent to the SNNI-CP/SNNI-CSP for  $dTA$  and by Theorem 6, it follows that CSNNI-CP and CSNNI-CSP are EXPTIME-complete. ■

Moreover, for  $dTA$ , thanks to the algorithm of section V there always exists a most permissive controller for CSNNI. However we will now show that there is a non-deterministic finite automaton s.t. there is no most permissive controller ensuring CSNNI.

**Proposition 4.** There is no most permissive controller ensuring CSNNI for the finite automaton  $A \notin dTA$  of figure 5(a) (i.e. such that  $A \setminus \Sigma_h$  is non deterministic) with  $\Sigma_h = \{h\}$ ,  $\Sigma_l = \{\ell_1, \ell_2, \ell_3\}$  and  $\Sigma_c = \{\ell_2, \ell_3\}$ .

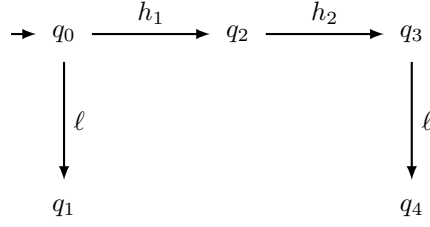
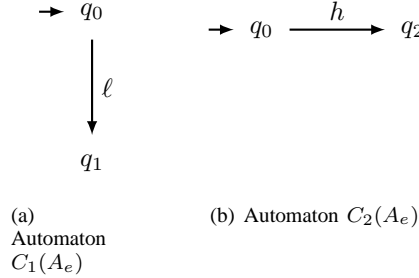
*Proof:*

Let  $A_c$  be the finite automaton of figure 5(a) with  $\Sigma_h = \{h\}$ ,  $\Sigma_l = \{\ell_1, \ell_2, \ell_3\}$  and  $\Sigma_c = \{\ell_2, \ell_3\}$ .  $A_c \notin dTA$  since  $A_c \setminus \Sigma_h$  is non-deterministic. This automaton is not CSNNI. The controllers  $C_1$  and  $C_2$  of figure 13 make the system CSNNI. However  $(C_1 \cup C_2)(A_c) = A_c$  is not CSNNI and, by construction is the only possible controller more permissive than  $C_1$  and  $C_2$ . Therefore, there is no most permissive controller ensuring CSNNI for  $A_c$  with  $\Sigma_c$ . ■

### B. BSNNI-CP and BSNNI-CSP

We first show by example 11 that even if there exists a controller for a finite automaton  $A$  and a controllable alphabet  $\Sigma_c$  ensuring BSNNI (i.e. the answer to BSNNI-CP is true), it is possible to have  $A \setminus \Sigma_c$  not BSNNI.

**Example 11.** Let  $A_i$  be the finite automaton of figure 14 with  $\Sigma_h = \{h_1, h_2\}$  et  $\Sigma_l = \{\ell\}$ . This automaton is BSNNI, then the answer to BSNNI-CP is true for all  $\Sigma_c$ . However, for  $\Sigma_c = \{h_2\}$ , the automaton  $A_i \setminus \Sigma_c = A_e$  is not BSNNI.

Fig. 14. The automaton  $A_i$ Fig. 15. Automata  $C_1(A_e)$  and  $C_2(A_e)$ 

We will now prove that for deterministic finite automaton there is not always a most permissive controller that enforces BSNNI. This result is in contrast with CSNNI where a most permissive controller always exists for dTA.

**Proposition 5.** *There is no most permissive controller ensuring BSNNI for the deterministic finite automaton of figure 6(a) with  $\Sigma_h = \{h\}$ ,  $\Sigma_l = \{\ell\}$  and  $\Sigma_c = \{\ell, h\}$ .*

*Proof:*

Let  $A_e$  be the deterministic finite automaton of figure 6(a) with  $\Sigma_h = \{h\}$ ,  $\Sigma_l = \{\ell\}$  and  $\Sigma_c = \{\ell, h\}$ . This automaton is not BSNNI. The controllers  $C_1$  and  $C_2$  of figure 15 make the system BSNNI. However,  $(C_1 \cup C_2)(A_e) = A_e$  is not BSNNI and, by construction is the only possible controller more permissive than  $C_1$  and  $C_2$ . Therefore, there is no most permissive controller ensuring BSNNI for  $A_e$  with  $\Sigma_c$ . ■

	A Timed Automaton		A Finite Automaton	
	$A \setminus \Sigma_h$ Non-Det.	$A \setminus \Sigma_h$ Det.	$A \setminus \Sigma_h$ Non-Det.	$A \setminus \Sigma_h$ Det.
CSNNI-CP	open	EXPTIME-C (Theorem 11)	PTIME (Theorem 9)	PTIME (Theorem 9)
CSNNI-CSP	NMPC* (Proposition 4)	EXPTIME-C (Theorem 11)	NMPC* (Proposition 4)	PSPACE-C (Theorem 10)
BSNNI-CSP	NMPC* (Proposition 5)	NMPC* (Proposition 5)	NMPC* (Proposition 5)	NMPC* (Proposition 5)

\* NMPC means that there not always exists a most permissive controller.

TABLE IV  
SUMMARY OF THE RESULTS FOR CSNNI AND BSNNI CONTROL PROBLEMS

The summary of the results for CSNNI and BSNNI Control Problems is given in Table IV.

## VII. CONCLUSION AND FUTURE WORK

In this paper we have studied the strong non-deterministic non-interference control problem and control synthesis problem in the timed setting. The main results we have obtained are: (1) the SNNI-CP can be solved if  $A \setminus \Sigma_h$  can be determinized and is undecidable otherwise; (2) the SNNI-CSP can be solved by solving a finite sequence of safety games if  $A \setminus \Sigma_h$  can be determinized; (3) there is not always a least restrictive (most permissive) controller for (bi)simulation based non-interference even for untimed finite automata. However, there is a most permissive controller for CSNNI if  $A \setminus \Sigma_h$  is deterministic and CSNNI-CP and CSNNI-CSP are EXPTIME-complete in this case in the timed setting.

The summary of the results is given in Tables I and II for the verification problems and Tables III and IV for the control problems.



Our future work will focus on the CSNNI-CP (and BSNNI-CP) as even when there is no most permissive controller it is interesting to find one. Another future direction will consist in determining conditions under which a least restrictive controller exists for the BSNNI-CSP.

## REFERENCES

- [1] R. Focardi, R. Gorrieri, Classification of security properties (part I: Information flow), in: R. Focardi, R. Gorrieri (Eds.), *Foundations of Security Analysis and Design I: FOSAD 2000 Tutorial Lectures*, Vol. 2171 of *Lecture Notes in Computer Science*, Springer-Verlag, Heidelberg, 2001, pp. 331–396.
- [2] A. Sabelfeld, A. Myers, Language-based information-flow security, *IEEE Journal on Selected Areas in Communications* 21 (1) (2003) 1–15.
- [3] R. Focardi, R. Gorrieri, The compositional security checker: A tool for the verification of information flow security properties, *IEEE Trans. Softw. Eng.* 23 (9) (1997) 550–571.
- [4] R. Focardi, A. Ghelli, R. Gorrieri, Using non interference for the analysis of security protocols, in: *Proceedings of DIMACS Workshop on Design and Formal Verification of Security Protocols*, 1997.
- [5] A. Bossi, C. Piazza, S. Rossi, Compositional information flow security for concurrent programs, *J. Comput. Secur.* 15 (3) (2007) 373–416.
- [6] G. Barthe, D. Pichardie, T. Rezk, A certified lightweight non-interference java bytecode verifier, in: *Proceedings of the 16th European conference on Programming, ESOP'07*, Springer-Verlag, 2007, pp. 125–140.
- [7] F. Kammüller, Formalizing non-interference for a simple bytecode language in coq., *Formal Asp. Comput.* 20 (3) (2008) 259–275.
- [8] M. Krohn, E. Tromer, Noninterference for a practical difc-based operating system, in: *Proceedings of the 2009 30th IEEE Symposium on Security and Privacy, SP '09*, IEEE Computer Society, Washington, DC, USA, 2009, pp. 61–76.
- [9] R. van der Meyden, C. Zhang, Algorithmic verification of noninterference properties, in: *Proceedings of the Second International Workshop on Views on Designing Complex Architectures (VODCA 2006)*, Vol. 168 of *Electronic Notes in Theoretical Computer Science*, Elsevier, 2006, pp. 61–75.
- [10] D. D'Souza, K. R. Raghavendra, B. Sprick, An automata based approach for verifying information flow properties, *Electr. Notes Theor. Comput. Sci.* 135 (1) (2005) 39–58.
- [11] A. Saboori, C. Hadjicostis, Opacity-enforcing supervisory strategies for secure discrete event systems, in: *the 47th IEEE Conference on Decision and Control*, 2008.
- [12] F. Cassez, J. Dubreil, H. Marchand, Dynamic Observers for the Synthesis of Opaque Systems, in: *7th Int. Symp. on Automated Technology for Verification and Analysis (ATVA'09)*, Vol. 5799 of *Lecture Notes in Computer Science*, 2009, pp. 352–367.
- [13] F. Cassez, J. Dubreil, H. Marchand, Synthesis of opaque systems with static and dynamic masks, *Formal Methods in System Design* 40 (1) (2012) 88–115.
- [14] F. Cassez, The Dark Side of Timed Opacity, in: *Proc. of the 3rd International Conference on Information Security and Assurance (ISA'09)*, Vol. 5576 of *Lecture Notes in Computer Science*, Copyright Springer, Seoul, Korea, 2009, pp. 21–30.
- [15] F. Cassez, J. Mullins, O. H. Roux, Synthesis of non-interferent systems, in: *4th Int. Conf. on Mathematical Methods, Models and Architectures for Computer Network Security (MMM-ACNS'07)*, Vol. 1 of *Communications in Computer and Inform. Science*, Copyright Springer, 2007, pp. 307–321.
- [16] Y. Moez, F. Lin, N. Ben Hadj-Alouane, Modifying security policies for the satisfaction of intransitive non-interference, *IEEE Transactions on Automatic Control* 54 (8) (2009) 1961–1966.
- [17] G. Gardey, J. Mullins, O. H. Roux, Non-interference control synthesis for security timed automata, in: *3rd International Workshop on Security Issues in Concurrency (SecCo'05)*, *Electronic Notes in Theoretical Computer Science*, Elsevier, San Francisco, USA, 2005.
- [18] G. Benattar, F. Cassez, D. Lime, O. H. Roux, Synthesis of Non-Interferent Timed Systems, in: *Proc. of the 7th Int. Conf. on Formal Modeling and Analysis of Timed Systems (FORMATS'09)*, Vol. 5813 of *Lecture Notes in Computer Science*, Budapest, Hungary, 2009, pp. 28–42.
- [19] R. Alur, D. Dill, A theory of timed automata, *Theoretical Computer Science* 126 (1994) 183–235.
- [20] O. Finkel, On decision problems for timed automata, *Bulletin of the European Association for Theoretical Computer Science* 87 (2005) 185–190.
- [21] O. Maler, A. Pnueli, J. Sifakis, On the synthesis of discrete controllers for timed systems, in: *STACS '95*, 1995.
- [22] D. D'Souza, P. Madhusudan, Timed control synthesis for external specifications, in: *STACS'02*, Vol. 2285 of *LNCS*, Springer, 2002, pp. 571–582.
- [23] L. J. Stockmeyer, A. R. Meyer, Word problems requiring exponential time: Preliminary report, in: *STOC*, ACM, 1973, pp. 1–9.
- [24] F. Laroussinie, P. Schnoebelen, The state-explosion problem from trace to bisimulation equivalence, in: *Foundations of Software Science and Computation Structures (FoSSaCS 2000)*, Vol. 1784 of *Lecture Notes in Computer Science*, Springer-Verlag, 2000, pp. 192–207.
- [25] K. Čerāns, Decidability of bisimulation equivalence for parallel timer processes, in: *Proceedings of the Fourth Workshop on Computer-Aided Verification, LNCS*, 1992.
- [26] S. Tasiran, R. Alur, R. P. Kurshan, R. K. Brayton, Verifying abstractions of timed systems, in: U. Montanari, V. Sassone (Eds.), *CONCUR*, Vol. 1119 of *Lecture Notes in Computer Science*, Springer, 1996, pp. 546–562.
- [27] T. Henzinger, P. Kopke, Discrete-time control for rectangular hybrid automata, in: *ICALP '97*, 1997.